AD-A053 013 SPERRY UNIVAC ST PAUL MINN DEFENSE SYSTEMS DIV F/G 9/2
RECONFIGURABLE COMPUTER SYSTEM DESIGN FACILITY INITIAL DESIGN S--ETC(U)
JAN 78 D R ANDERSON, L D ANDERSON, K Y WEN F30602-76-C-0355

UNCLASSIFIED RADC-TR-78-6-VOL-2 NL

1 OF 3
ADA
053013

2

RADC-TR-78-6, Volume II (of two)
Final Technical Report
January 1978

RECONFIGURABLE COMPUTER SYSTEM DESIGN FACILITY INITIAL DESIGN STUDY, Technical Results

D. R. Anderson
L. D. Anderson
K. Y. Wen

Approved for public release; distribution unlimited.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-78-6, Volume II (of two) has been reviewed and is approved for publication.

APPROVED: Michael A. Troutman

MICHAEL A. TROUTMAN, 2Lt, USAF
Project Engineer

APPROVED: Wendall C. Bauman

WENDALL C. BAUMAN, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISCA) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

See AD052995

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER (18)
RADC-TR-78-6, Vol II (of two) ✓

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER (9)

4. TITLE (and Subtitle) (6)
RECONFIGURABLE COMPUTER SYSTEM DESIGN FACILITY
INITIAL DESIGN STUDY, Technical Results.
Volume II.

5. TYPE OF REPORT & PERIOD COVERED
Final Technical Report,
30 Jun 76 - 30 Jul 77,

6. PERFORMING ORG. REPORT NUMBER
N/A

7. AUTHOR(s) (10)
D. R./Anderson,
L. D./Anderson
K. Y./Wen

8. CONTRACT OR GRANT NUMBER(s) (15)
F30602-76-C-0355 ✓

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Sperry Univac Defense Systems ✓
Univac Park, P.O. Box 3525
St. Paul MN 55165

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
62702F (17) 14
(16) 55971408

11. CONTROLLING OFFICE NAME AND ADDRESS
Rome Air Development Center (ISCA)
Griffiss AFB NY 13441

12. REPORT DATE (11)
Jan[redacted]978

13. NUMBER OF PAGES
261

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Same (12) 274p.

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: 2Lt Michael A. Troutman/ISCA

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Reconfigurable Computer System Design Facility, Total System Design Concept, System, Architecture, Total System Design Facility, Emulation, Networks, Performance Measurement, Design Languages

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

As a method of reducing design and development costs for data processing systems (hardware and software) a total system design concept is proposed. The concept includes a Reconfigurable System Design Facility (RCSDF) where total system design alternatives can be emulated for the purpose of evaluating and proving designs prior to actual development.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

408125

CONTENTS

Paragraph		Page
1	Introduction	1
1.1	Project Scope	1
1.2	Recommendations	4
1.3	Document Scope	5
1.4	Definitions	6
2	TSDF	8
2.1	TSDC/TSDF Evaluation	8
2.2	Evaluation of the Total System Design Concept	9
2.2.1	TSDC as a Part of the System Development Cycle	9
2.2.2	As a Complete Concept	14
2.2.3	TSDF Functional Capabilities	18
2.3	TSDF Functional Design Issues	31
2.3.1	TSDF Global Issues	32
2.3.2	TSDF Functional Model	35
2.4	A Viable TSDF Overview	39
2.4.1	Procedural Scenario	40
2.5	TSDF Advantages and Risk Assessments	57
2.5.1	TSDF Advantages	57
2.5.2	Risk Assessments of TSDF Functional Capabilities	62
2.6	Additional TSDF Issues and Recommendations	74
2.6.1	One TSDF vs. Many	74
2.6.2	Emulation as an Aspiration	75
2.6.3	Hosted Capabilities	76
2.6.4	Users	76
2.6.5	System vs. Machine Architecture	77
2.6.6	Performance	77
2.6.7	User Insight into the TSDF	78
3	Reconfigurable Computer System Design Facility (RCSDF)	79
3.1	Fundamental RCSDF Doctrines	79
3.1.1	Emulation vs. Architecture Development	79
3.1.2	Research Vehicle	80
3.1.3	User Accessibility to Equipment	80
3.1.4	System vs. Data Processing	80
3.1.5	Hosting	80

CONTENTS

Paragraph | Page

3.1.6 RCSDF Operating Environment 81
3.2 Emulation Operating Philosophy 81
3.2.1 System Formulation 83
3.2.2 Scenario/Test Environment; System Testing 92
3.3 RCSDF Facility Capabilities 97
3.3.1 System Design 98
3.3.2 System Testing 106
3.4 RCSDF Emulation Procedures 119

4 RCSDF Development Plan 125
4.1 Definitive Study Tasks 125
4.1.1 Emulation System Architecture 125
4.1.2 Emulation Control Structure 130
4.1.3 Emulation Documentation Structure 132
4.1.4 Requirements/Design Language Procedure 133
4.1.5 Uniform Emulation Method 136
4.1.6 Emulation Analysis Structure 137
4.2 Developmental Paths 140
4.2.1 RCSDF Hardware 140
4.2.2 RCSDF Software 140
4.2.3 Emulation Procedures 141
4.2.4 Host System Software 141
4.2.5 Design Languages 142
4.2.6 Case Study/Training 142
4.3 Description of RCSDF Development Tasks 143
4.3.1 Emulation System Architecture Study 143
4.3.2 Emulation Control Structure Study 143
4.3.3 Emulation Documentation Structure Study 144
4.3.4 Requirements/Design Language Procedural Study 144
4.3.5 Uniform Emulation Study 144
4.3.6 Emulation Analysis Structure Study 144
4.3.7 Additional Hardware Specifications 145
4.3.8 Additional Hardware Procurement/ Modification and Testing 145
4.3.9 Hardware Facility Control Specifications 145
4.3.10 Facility Control Hardware Procurement and Testing 145

CONTENTS

Paragraph		Page
4.3.11	Interface Standardization Study	146
4.3.12	Communication Protocol Development	146
4.3.13	Facility Component Interface Specifications	146
4.3.14	Facility Component Interface Developments	146
4.3.15	Facility-Host Interface Specification and Development	147
4.3.16	Peripheral Interface Specification and Development	147
4.3.17	Deployable Component Interface Specification	147
4.3.18	Peripheral Hardware Procurement	147
4.3.19	Performance Monitoring Hardware Specifications	147
4.3.20	Performance Monitoring Hardware Procurement	148
4.3.21	RCSDF Hardware Integration Testing	148
4.3.22	Procurement Procedures Study	148
4.3.23	Responsibility Delineation Description Study	148
4.3.24	RCSDF Structural Description	148
4.3.25	Application Selections for Case Studies	149
4.3.26	Requirement Specifications Case Studies	149
4.3.27	Functional Decomposition Case Studies	149
4.3.28	Hardware Implementation Case Studies	150
4.3.29	Software Implementation Case Studies	150
4.3.30	Emulation Driver/Procedures Case Studies	150
4.3.31	User Manual Preparation	150
4.3.32	Facility Control Software Specifications	150
4.3.33	Facility Control Software Procurement and Testing	151
4.3.34	Support Software Specifications	151
4.3.35	Support Software Procurement and Testing	151
4.3.36	Emulation Performance Measurement Software Specification	151
4.3.37	Emulation Performance Measurement Procurement and Testing	152
4.3.38	Support Library Specifications	152
4.3.39	Simulation Software Specification	152
4.3.40	Simulation Software Procurement and Testing	153

CONTENTS

Paragraph		Page
4.3.41	RCSDF Software Integration Testing	153
4.3.42	Facility Data Base Management System (DBMS) Specifications	153
4.3.43	Facility DBMS Procurement and Testing	154
4.3.44	Host Data Base Tool Specification	154
4.3.45	Host DB Tools Procurement and Testing	154
4.3.46	Static Initialization Software Specification	154
4.3.47	Static Initialization Software Procurement and Testing	155
4.3.48	Language Translators Procurement and Testing	155
4.3.49	Host Related Design Language Specifications	155
4.3.50	RCSDF Related Design Language Specifications	156
4.4	RCSDF Work Breakdown Structure	157
4.5	Time Phased RCSDF Development Plan	157
5	RCSDF Baseline Studies	163
5.1	Performance Measurement Technical Baseline	163
5.1.1	Monitoring Techniques	163
5.1.2	Data Presentation Techniques	168
5.1.3	RCSDF Performance Measurement Requirements and Recommendations	169
5.2	Processor Communication Techniques/ Protocol Technical Baseline	171
5.2.1	Intercomputer Processor Communication	173
5.2.2	Networks	177
5.2.3	Arrays and Ensembles	182
5.2.4	RCSDF Processor Communications Recommendations	183
5.3	Microprocessor Network Technical Baseline	187
5.3.1	Microprocessor - Survey and Assessment	188
5.3.2	Interprocessor Connections - Survey and Assessment	191
5.3.3	RCSDF Microprocessor Network Requirements and Recommendations	197
5.4	Microprogramming Technical Baseline	200

CONTENTS

Paragraph		Page
5.4.1	Microprogramming Overview	200
5.4.2	Applications of Microprogramming	203
5.4.3	Microprogrammed Architectures	206
5.4.4	Software Aids	208
5.4.5	RCSDF Emulation	209
5.5	Operating System Technical Baseline	213
5.5.1	Trade-Off Evaluations	213
5.5.2	Special RCSDF Operating System Requirements	222
5.5.3	RCSDF Operating System Recommendations and Conclusions	223
5.6	Distributed Systems Organization Technical Baseline	226
5.6.1	Remote Network Systems (RNS)	227
5.6.2	Local Network Systems (LNS)	229
5.6.3	Multiprocessors	231
5.6.4	Distributed Function Systems (DFS)	233
5.6.5	Distributed Element Systems (DES)	234
5.6.6	RCSDF Requirements in Distributed Systems	236
5.6.7	RCSDF Distributive Systems Organization/Recommendations/Conclusions	237
5.7	Design Languages Technical Baseline	239
5.7.1	Language Identification and Evaluation	239
5.7.2	RCSDF Design Language Requirements	251
5.7.3	RCSDF Design Language Recommendations and Conclusions	254
6	Conclusions	255

LIST OF FIGURES

Figure		Page
2-1	Total System Design Concept	15
2-2	Total System Design Concept (Modified)	18
2-3	TSDF Functional Model	36
2-4	A TSDF	41
2-5	Example of PDL	45
2-6	An Adaptation of ISP	46
2-7	An AN/UYK-20 Instruction Description in ISP	46
2-8	Example of a Hosting Subsystem	48
3-1	Responsibility Delineation	84
3-2	RCSDF Emulation Procedures	120
4-1	Network	128
4-2	Mainframe	128
4-3	Sequential Emulator	129
4-4	Parallel Processor	129
4-5	Hierarchical System	129
4-6	Relationship of Performance Measurement Units	139
4-7	Work Breakdown Structure	158
4-8	Time Phased Plan	161
5-1	Distributed Systems Taxonomy	228
6-1	RCSDF Development Plan Time Lines	256

LIST OF TABLES

Table		Page
4-1	Task Index	159

1. INTRODUCTION

The costs of developing software for modern military systems are increasing at alarming rates. Awareness of this situation has motivated an exploration of the causes of this cost increase and ways to reduce it. One alternative that shows significant promise for reducing future software costs (and ultimately total system costs) is total system design, which requires system design to be completely developed and evaluated within a design host environment (i.e., a Total System Design Facility - TSDF) especially created to provide the necessary tools, evaluation techniques, and methods needed for extensive monitoring of the total development process. The total system design concept envisions a disciplined system design environment that allows design alternatives to be quickly and easily evaluated and the overall system design to be thoroughly examined to assure conformance with prescribed cost/performance profiles. In addition, having a facility with the ability to test a total system design concept prior to the actual development of the system will minimize "fielded-faults" which are a major cause of increased system life cycle costs in newly deployed systems.

1.1 Project Scope

The Reconfigurable Computer System Design Facility (RCSDF) initial design study was commissioned to perform technology studies in support of the development of a facility that would be capable of supporting research into the concept of total system design. The objective of the initial RCSDF design study was the preparation of a development plan describing the necessary studies and development tasks that would achieve the required facility capabilities.

The RCSDF initial design study was organized into three major tasks:

Task 1 RCSDF Capabilities, Philosophy, Procedures

- o Evaluation of total system design facility concept
- o Definition of capabilities, philosophy, and procedures for the reconfigurable computer system design facility

Task 2 RCSDF Technical Baseline Development

- o State-of-the-art technical baseline studies in
  - Performance measurement
  - Processor communications techniques/protocol
  - Microprocessor network
  - Operating system
  - Microprogramming
  - Distributed systems organizations
  - Design languages

Task 3 RCSDF Development Plan

- o Development threads
- o Time-phased development tasking
- o Work breakdown structure

The three tasks of the nine-month RCSDF initial design study were performed sequentially starting from the TSDF concept and ending with a four-year development plan defining the paths of development and time-phased tasks necessary to achieve a 1980s demonstration of the TSDF concept with available hardware and technology -

the RCSDF.

The initial RCSDF design study was staffed with highly experienced personnel who have demonstrated proficiency in advanced system design:

D. R. Anderson, B.S.E.E., Project Engineer
K. J. Thurber, Ph.D. Electrical Engineering
L. D. Anderson, M.S. Mathematics
K. Y. Wen, Ph.D. Computer Science
H. A. Freeman, Ph.D. Electrical Engineering
D. O. Schneider, Senior Programmer
R. A. Eggan, Senior Applications Analyst

The initial RCSDF design study was performed using the philosophy of an independent appraisal of the TSDF concept, performance of baseline studies, and preparation of a development plan. Periodic technical interchanges with Rome Air Development Center personnel were used to guide the study direction and to provide updated insight into the RCSDF development progress. The formative nature of the RCSDF development was recognized at the onset of the study and this necessitated that the study proceed on a broad basis, examining fundamental technology and computing techniques in preparation for planning the RCSDF technical development. It was also recognized that other DoD efforts were also examining facets of the software cost problem, so that when defining RCSDF development needs an attempt was made to minimize effort duplication. However, total success in eliminating duplication was probably not achieved since visibility of the entire DoD activity in the software development area was limited by the prescribed time-scope of the contract and the multitude of DoD efforts to reduce software costs.

1.2 Recommendations

In general, the Sperry Univac study team has found RADC's concept of total system design utilizing the reconfigurable computer system design facility for system evaluation to be a viable method with significant potential for reducing future system hardware and software costs. In order to avoid the potential time-lag required to master the evaluation capabilities of the RCSDF, the Sperry Univac study team recommends, via the proposed development plan, that RADC emphasize the following tasks for RCSDF development in the near future (12 to 18 months):

Emulation System Architecture (Paragraph 4.3.1)

Formulation of RCSDF architecture descriptions, their evaluation with respect to different user problems, and recommendation of a specific architecture as a research evaluation vehicle.

Emulation Control Structure (Paragraph 4.3.2)

Definition of interface standards to regiment resource control for the emulation system.

Emulation Analysis Structure (Paragraph 4.3.6)

Description of how process control philosophy can be utilized for performance measurement/analysis; functional specification of the components (processes) required as a part of the RCSDF configuration.

The Sperry Univac study team further recommends that the tasks identified in the proposed development plan (Section 4.0) and tailored to meet a specific applications case study be implemented to provide RADC with the most timely benefits in demonstrating the total system design concept at lower risk and cost. In arriving at these recommendations technical risks have been identified for a general purpose emulation facility (Section 2.5). Sperry Univac has concluded that to achieve a comprehensive emulation facility, the TSDF fully supported with services would require many years of development, and the use of such a facility would not be possible for near-to-mid term programs. However, by pragmatically limiting the scope of the program and accepting moderate degrees of inefficiency we can achieve useful results in the near-to-midterm. The RCSDF case study development plan alternative (Figure 6-1c) is recommended for near-term RCSDF development.

1.3 Document Scope

The technical information developed during the initial RCSDF design study is contained in the following:

Volume I	Initial RCSDF Design Study Technical Summary
Volume II	Initial RCSDF Design Study Technical Results
Volume III	Initial RCSDF Design Study Source Documentation

Volumes I and II comprise the final report of the efforts performed under the subject contract. Included in these volumes is material describing the TSDF concept and its assessment, a description of the RCSDF and its evaluation procedures, a recommended RCSDF development plan, and summaries of the technical baseline studies performed to support the development plan.

1.4 Definitions

Terms used freely throughout this document which may be unfamiliar to the reader are:

<u>RCSDF</u> Reconfigurable Computer System Design Facility
The RCSDF is a specific computer facility at Rome Air Development Center to be used to evaluate the concept of total system design.

<u>TSDC</u> Total System Design Concept
The abstract process by which a total system design is developed and evaluated.

<u>TSDF</u> Total System Design Facility
The facility and procedures supporting the operation of the facility that implements the TSDC.

<u>HOL</u> Higher Order Language
A programming language that allows "English-like" constructs, such as FORTRAN, PL/I, COBOL, and JOVIAL.

<u>HLHDL</u> High Level Hardware Design Language
A programming language comparable to HOL, used for the purpose of describing hardware logic designs.

EDL Emulation Design Language
A HLHDL, specifically used for emulation design analysis.

2. TSDF

The TSDF is a conceptual facility incorporating methods for designing and evaluating computer architecture design alternatives. The conceptual TSDF was evaluated to determine its advantages as a system design technique and the extent to which it would be useful in the system design process. Risks in achieving a TSDF capability were also examined.

2.1 TSDC/TSDF Evaluation

After evaluating the TSDC and its implementation, the TSDF, the following conclusions were reached:

- The TSDC, as presented, has a wider applicability than the development of hardware and software design specifications. Its scope could extend into requirements formulation phases at the start of development, system tuning, and software development during the early phases of procurement.
- The TSDC, as presented, lacks an adequate capability for analytic determination of operating performance. Tools should be provided for users to guide the direction of their development. Such tools would be used to provide only approximate answers.
- More emphasis should be placed upon specifying and implementing the digital processing environment under which the emulated/simulated system is being tested.
- Hosting of the generation process (system emulation/simulation, environment, etc.) is an essential ingredient of the TSDF. The anticipated complexity of testing, the possible multiplicity of users, and the desire to minimize the transit time from conception to specification all point to the need to centralize system generation using a data

base.

- o The definition of a High Level Performance Measurement Language (HLPML) ought to be pursued as a part of the TSDF to provide users with a capability comparable to that in the HOL, HLHDL.
- o The TSDF should be regarded as a parent and siblings, with the parent assuming broad scale logical capabilities and the siblings responsible for proving performance.

2.2 Evaluation of the Total System Design Concept

The Total System Design Concept (TSDC) is examined in this section from two points of view:

1) as part of the system development cycle
2) as a complete concept

The first point of view, in which the TSD fits into the Air Force development cycle, reflects what should be the natural boundaries of the concept. If a TSDF is envisioned, limitations imposed should occur on "natural" system development boundaries or duplication of system development efforts will occur. The second point of view looks at the TSDC in terms of completeness. Is there anything missing? Is the TSDC methodology clear? The functional capabilities of a TSDF are then described to charaterize the TSDF scope of operations.

2.2.1 <u>TSDC as a Part of the System Development Cycle</u> - Conceptually, systems originate from statements of operational needs. These may be in the form of Required Operational Capability (ROC) or Technical Directive (TD) statements that concisely describe the desired capability but do not define how that capability is achieved. From capability statements, requirements are evolved, and these requirements are formulated into systems designs which proceed through stages of definition and development until final procured configurations are arrived at. TSDC is postulated as starting from "problem" (interpreted to mean requirements) and leading directly to software design specifications. However, we know from experience that requirements are not always defined and quantified precisely enough to enable software or hardware specifications to be made immediately. Research in BMD (Ballistic Missile Defense) technology has attempted to solve some of the problems associated with translating operational needs to a tested real-time process.[1]

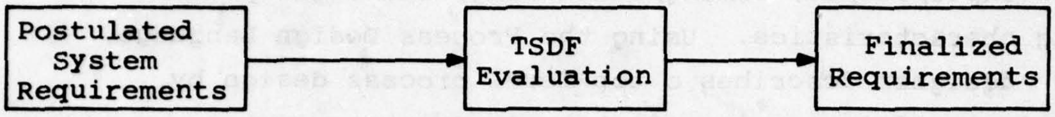
> The Process Design System is an integrated package of software development tools accessed through a single Process Design Language. These tools---translators, library/configuration management, simulation tools, data collection report generation programs, process models, etc.--are invoked to develop an experimental process subject to execution in a simulated environment which is also defined and controlled by the Process Design Language.
>
> The methodology combines the complementary disciplines of top-down design, structured programming, and evolutionary development with advanced construction techniques to permit functional and analytic (or a combination of) process modules to be executed in a simulated environment, either in real time or interrupted real time.

[1] BMD Advanced Technology Center, "BMDATC Software Development System - Program Overview " I (July 1975).

Based on a requirements analysis, processing tasks are identified and grouped into functional units with common input/output, timing/scheduling, and logic processing characteristics. Using the Process Design Language, the designer describes a top-level process design by assigning functional units to appropriate computer resources, establishing communication requirements, and defining scheduling rules for real time control and synchronization. The resulting functional model of the process is exercised against a functional simulation of the operational environment.

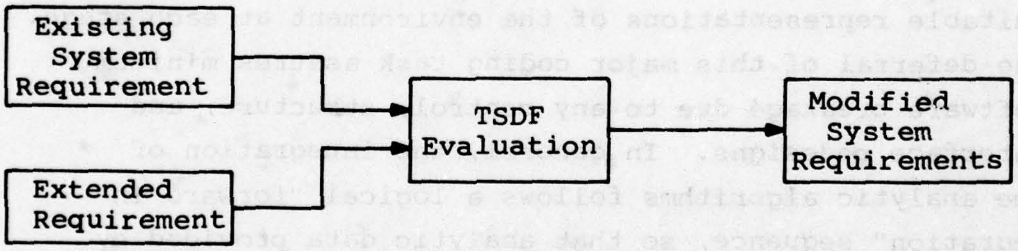
Further decomposition establishes the control structure or sequencing logic. As the actual code modules become available, they are inserted into the control structure, replacing their functional counterparts. The hybrid of functional and analytic modules remains executable against suitable representations of the environment at each stage. The deferral of this major coding task assures minimum software breakage due to any control, structure, and interface redesigns. In general, the integration of the analytic algorithms follows a logical "forward integration" sequence, so that analytic data provided by the initial processing steps is available for subsequent use by other analytic algorithms. Iteration of this implementation, test, and evaluation cycle results in the completed real-time process.

Definition of operational parameters such that a system is plausibly achievable with known technique is not included within the current definition of the TSDC. It is possible that the TSDF could be used to help define requirements parameters (tracking capacity bounds, for example); symbolically this can be represented as:



The attractiveness of this function is that feedback is provided to the system acquisition process that could impact procurement decisions; for example, how many systems to procure based upon known performance factors. Maybe in some instances one system would provide adequate performance, where originally two systems were thought to be necessary.

In a slightly different vein, consider an existing system whose requirements are modified to extend the original requirements set forth, as depicted below:



In this case, as before, the goal of TSDF is to determine if the modified system requirement is plausibly achievable. In some cases this could demonstrate the ability of existing equipment to meet extended requirements. In others it would indicate what the new system design should be (hopefully requiring only a subsystem replacement or addition).

The use of TSDF during system formulation (starting at defined requirements and ending with specifications) is obvious. However, since the TSDF hardware is not the real system hardware, TSDF involvement with system hardware seems naturally to end with the generation of the system hardware specification. Not so with software. If the TSDF hardware faithfully emulates (performance not considered), it seems that the TSDF could aid in the development of software during early stages of software procurement. Extension of the TSDF to certification may not be possible because of performance but logical operation of the software system (or subsystem) should be possible.

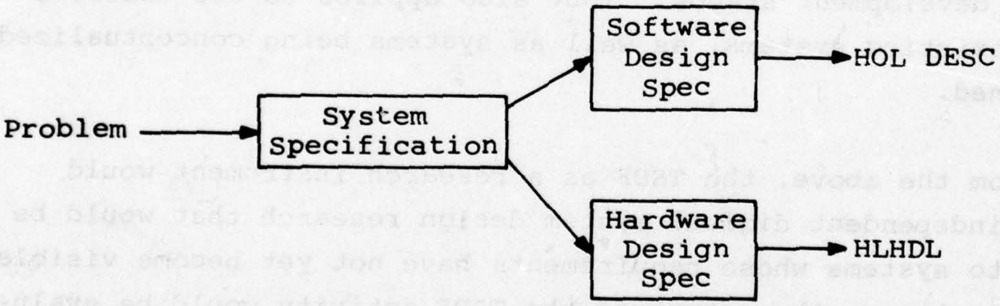
Summarizing, for cases where it is necessary to translate into systems designs, it is felt that the TSDF has wider applicability than originally portrayed. Its function could begin in the earlier systems requirements definition phases and extend to actual software development stages. TSDF also applies to the modification of existing systems, as well as systems being conceptualized and defined.

Apart from the above, the TSDF as a research instrument would support independent digital system design research that would be applied to systems whose requirements have not yet become visible. In this instance, the output of the TSDF activity would be evaluation data that provides insight into the effectiveness of alternate system architectural techniques. In the production of this evaluation data it can be expected that exploratory work would be performed to develop new techniques. Problems, or requirements in this instance, would be synthesized to provide a framework for technique evaluation. The importance of this role is not to be discounted, in that it provides a framework for postulating new approaches to meeting systems requirements.

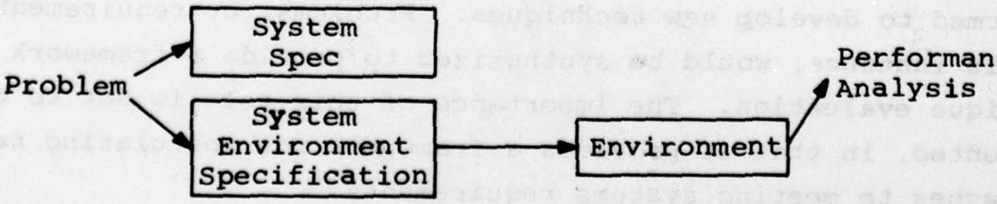
2.2.2 As a Complete Concept - The TSDC (and ultimately the usage of the TSDF) has been presented as shown in Figure 2-1. A problem is entered into the system and is reflected in a software design specification which leads to HOL-compiled target software. In parallel with this, an analogous path is followed to develop hardware architecture using High Level Hardware Design Language (HLHDL). The separate hardware and software paths become related at the point of code generation where hardware architecture is incorporated into the compiled code installed in the TSDF. In the performance analysis that follows both hardware and software are iterated until a satisfactory solution is reached.

Comments on concept completeness are as follows:

1. System Specification - since this is a system design concept perhaps the problem should lead to a system specification which would then lead to separate software/hardware design specifications.



In addition, an environment specification should be tied to the PROBLEM to provide a basis for judging performance:



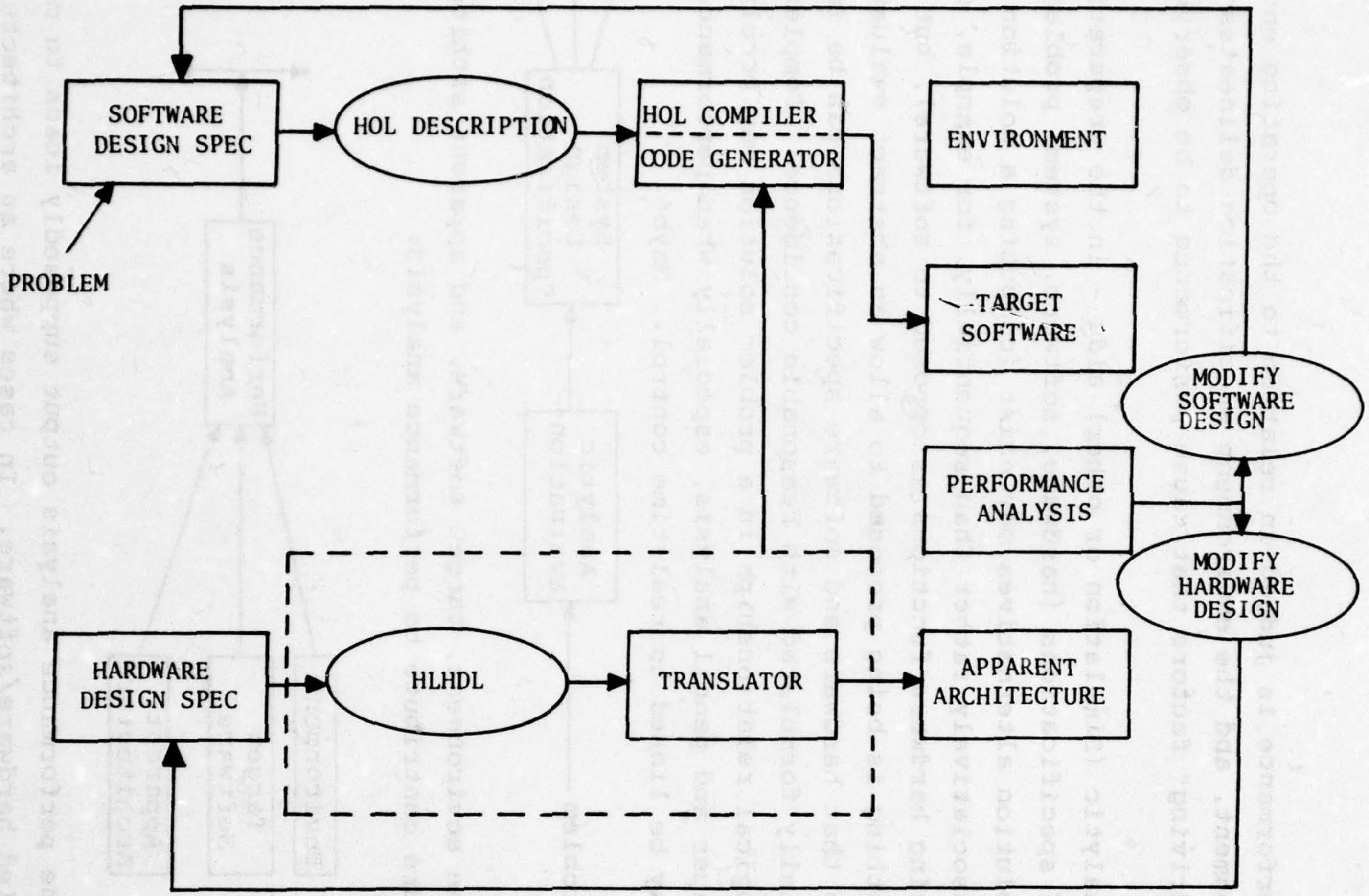
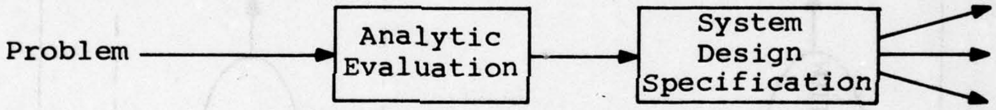


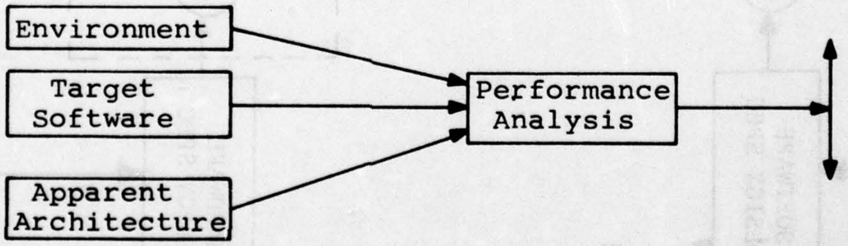
Figure 2-1. Total System Design Concept

Performance is judged in relation to the operating environment, and the environment specification delineates the "driving" factors that cause performance to be observed.

2. Analytic (Simulation or other) aids - in the preparation of specifications (hardware, software, system) problem solution alternatives may exist (computing a solution associatively rather than sequentially, for example, or using hardware functions as opposed to software), but nothing is being proposed to allow an abstract evaluation so that hardware and software specifications can be initially formulated with reasonable confidence. Complex logical relationships in a problem solution may preclude paper and pencil analysis, especially when performance may be linked to real-time control. Maybe:



3. The environment, target software, and apparent architecture contribute to performance analysis:



4. The performance analysis output supposedly leads to modified hardware/software. In cases where an architectural change (hardware/software form) appears to be needed, no assistance is provided to determine what these changes

should be. Again complex internal system interaction between parts may preclude determining the effect of changes made.

5. The output of the HLHDL translator to the apparent architecture impacts the environment - especially if environment must create a different form of input data to the system in response to an architectural change, if parallel data channels replaced serial channels, for example.

6. If a system specification phase is included in TSDC then performance analysis should lead back to the system specification and thence onto modified hardware/software specs.

The overall effect of comments 1 through 6 are depicted in Figure 2-2.

2.2.3 TSDF Functional Capabilities - Ideally, the TSDF should be perceived by the "user" as the target architecture with the internal emulation details totally masked. This ideal perception will be compromised with some users; namely, those whose interests lie with creating experimental or high performance architecture for which there are no standard components. For users who can rely on standard components, the ideal perception could be realized provided they have confidence in the transformation tools provided. To all appearances, the TSDF should be observed to logically behave as the target architecture. System performance (the observed performance modified by some transformation that factors out emulation overhead and performance differentials between components used and components intended to be used) is determined in nonsystem time (after the emulation has been performed) as an

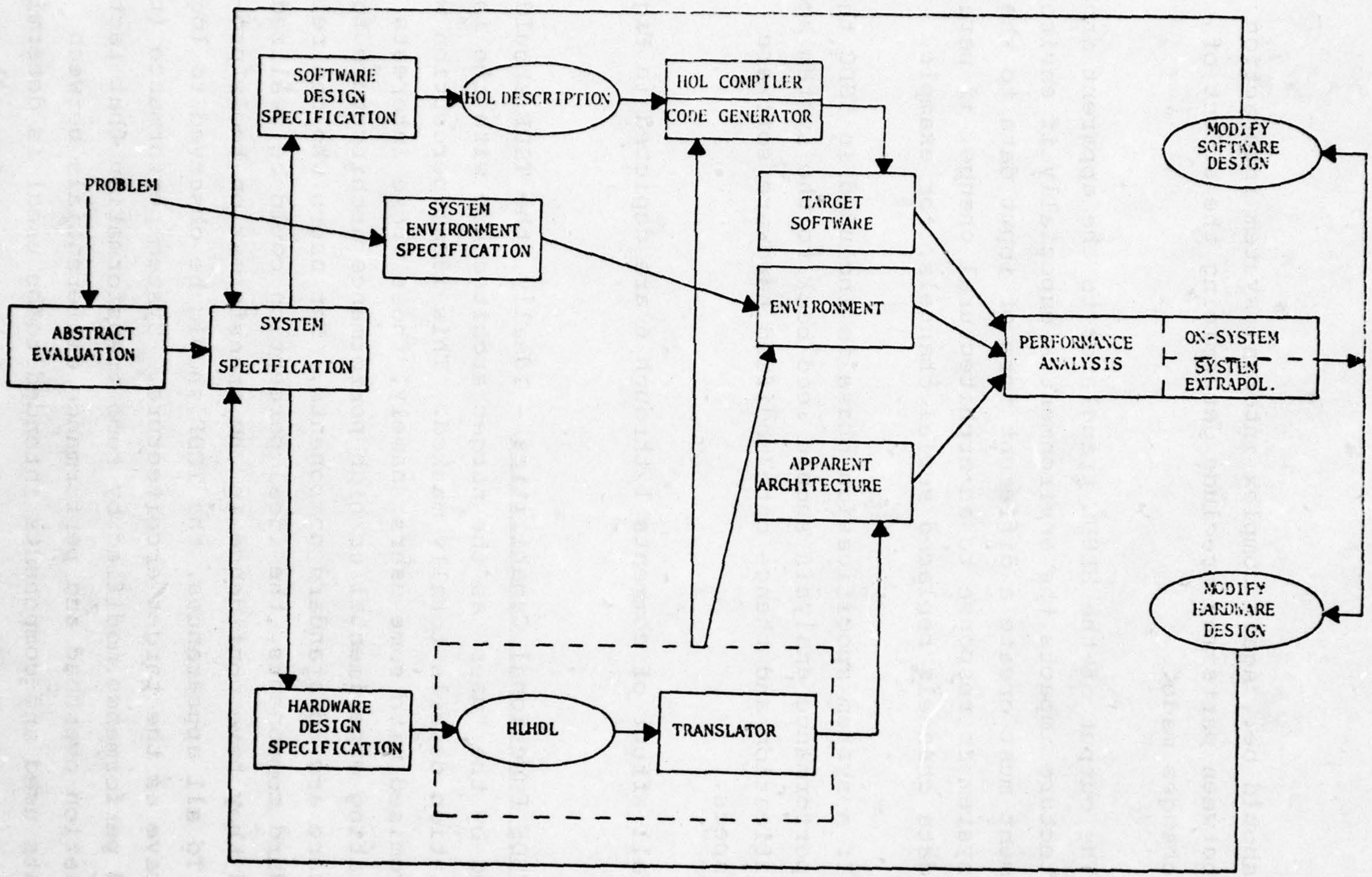


Figure 2-2. Total System Design Concept (Modified)

analysis function. A user should be able to logically interact with the TSDF as he would expect to with the emulated system but not in a device-specific manner. (He should not be required to interface with the TSDF through the specific interface devices of the planned system but may do so if it is relevant.)

2.2.3.1 System Environment - It seems logical to assume that a TSDF would be comprised of multiple computing elements. The system environment of the TSDF is the collection of features and characteristics presented to the user as a total facility package. Features, characteristics and capabilities of the TSDF environment are discussed below.

2.2.3.1.1 Simultaneous Hosting of Separate Users - The TSDF should have the functional capability of hosting only one user at a time. This means that users who have distinctly different architectures are not present on the TSDF simultaneously. A user who is emulating or simulating a multiprogramming environment, however, should be provided with the capability for doing so. If the single-user restriction is not invoked, then the TSDF operating system must support both program and system configuration state saving; performance measuring must also be multiprogrammed and be able to track program activation and deactivation. Multi-user, simultaneous usage of the TSDF creates unnecessary complications for the TSDF operating systems.

2.2.3.1.2 Realtime/Non-Real-time Operations - The TSDC does not appear to require that the TSDF operate at the expected performance level of the target system. The target system performance level is defined as being "real-time." Environments that drive the TSDF need only to simulate real-time inputs to the degree needed to realistically scale performance to the target system. Internal TSDF operations should be at performance levels which can be realistically scaled to target system operations. Defining this performance level is a definite problem. Operational performance levels which are vastly different from the desired target system performance may 1) be difficult to scale and 2) lack credibility and thus erode confidence in the developed specification.

2.2.3.1.3 Virtual Environment - The TSDF should have the functional capability to support two kinds of virtual environment. It should support a virtual environment that allows a user to host a process on any of several processing elements. It should also allow a user to host a virtual memory environment as part of an emulation/simulation. The first kind of virtual environment allows a user the freedom to examine different computing solutions to the same problem. The second kind of virtual environment allows the hosting of contemporary architecture and may or may not be hardware-aided.

2.2.3.1.4 Synthetic/Real System Environment - Simulated real system environments (synthetic environments) seem adequate for TSDF objectives. Duplication of real system inputs/outputs would require excessive attention, diluting the available resources needed for the major TSDF objective. Synthetic environments also have the advantage that crucial design parameters can be examined in isolation from unrelated aspects of the system design.

2.2.3.1.5 Dynamic Architecture Alteration - Users should not be permitted to reconfigure or alter the TSDF architecture as part of their on-system evaluation. They may elect to select alternative usage of the configured system but may not change the configured system in real-time, except as required by the target system emulation. To a large extent, this can be regulated by the method of generating the hardware configuration (HLHDL).

2.2.3.1.6 Existing and New Architecture - The TSDF should have the capability to host (simulate or emulate) both existing and new architectures. Architectural development is evolutionary; existing architecture forms a baseline from which to evolve. One may elect, for example, to add system components to an existing architecture in order to meet extended system requirements. In this case the existing architecture would be modified (by changing the HLHDL) to include the added components hosted on the TSDF, and its behavior would be analyzed. New, exploratory architecture is the subject of development in the TSDF.

2.2.3.1.7 Hosted System Preservation - The TSDC should have the capability of preserving all particulars of a hosted system design, including a capability for restarting the system from a termination point, should that be desired. The system components simulated may be placed in a data bank for recall as needed. This has advantages, in that after a system has been created through HLHDL, modifications would not seem to require an entire recompile, making restart of the system evaluation simpler.

2.2.3.1.8 Multiple Machines - The TSDF should have the capability to simulate the architecture of a multiple-machine system of either a homogeneous or heterogeneous nature. This includes the simulation of interconnection and multiple system input/output. The multiple-machine simulation capability must include simulating (or emulating) associative, parallel, and sequential processing.

2.2.3.1.9 Parallel Operations - Multiple machines of the TSDF must be capable of operating in parallel should the user desire it. The study of techniques to resolve concurrency problems for multiple machines (as a subset of the user problem set) will require this. Sequential operation of multiple machines can be treated as a subset of the parallel operations capability.

2.2.3.1.10 Control Methodology - The TSDF should be capable of mapping software components to hardware processor elements, either one-to-one or n-to-one, without redesign. This requires that the control methodology for the TSDF recognizes each software component as a stand-alone element. Such recognition is compatible with a unique application of finite-state machine theory which recognizes that the output of a software component (process) depends not only on the correct input but also on the entire history of inputs as summarized by the current state of the hardware element. This in turn, demands that the control methodology for the TSDF make available control algorithms for manipulating machine state vectors in order to

2.2.3.1.10 (continued)

control scheduling, resource assignment, resource sharing, and the accepted control hierarchy.

2.2.3.2 Processing Capabilities - Without the boundaries of a defined user class, the TSDF, conceptually speaking, must have the capabilities of simulating or emulating all known and unknown architectural techniques. Realistically, this is not achievable under economic or time constraints. Therefore limits must be established that will make the TSDF more manageable, i.e., configurations or users must be limited or alternatives must be provided.

The TSDF should have the functional capability of emulating/simulating all of the following processing capabilities, as a minimum:

1. Array
   - Parallel
   - Associative
2. Pipeline
3. Unit
4. Multiprocessor
5. Federated
6. Distributed
7. Functionally distributed

Further, the TSDF should have the functional capability to host these processing capabilities as either separate distinct machines, as distinct independent computing elements of a single machine, or as distinct processing capabilities of a unit machine. To date PEPE is the only known architecture assimilating all these alternatives.

The TSDF should have the capability to alter the characteristics of each of the processing capabilities: stages in a pipeline, CPUs comprising the multiprocessor, CPU addressing structures, etc. Emulated/simulated performance measurements should be extendable (scalable) to the target architecture through simulation.

In addition, the TSDF can work with processing components that are nearer in design to the components that are intended to be used in the target architecture. This would allow a more exact performance simulation.

2.2.3.3 Memory Capabilities - The TSDF memory system should be 1) structurable, 2) sharable, and 3) definable in traditional forms. Structurability allows the defining of primary, secondary, tertiary, and other levels that are commonplace in many existing system architectures and which form a cornerstone upon which processing concepts are based. Emulation of existing systems would require this capability. For multiprocessing and some distributed processing systems sharability of memory is required.

The TSDF memory forms should encompass byte, word, and block addressability and the capability to define memory operations and structure alternative to the existing forms to create new memory architectures. Further, the memory forms should be replaceable in the TSDF architecture with higher or alternate performance designs to bring TSDF performance nearer the target architecture performance.

For new architecture, memory should be constructable (in an emulation/simulation sense) with embedded processing functions.

2.2.3.4 Data Entry/Exit Interfaces - TSDF data entry/exit should be emulated/simulated in direct proportion to its importance in the target architecture (the architecture which is being emulated). If , for example, the target architecture is oriented toward an application that is dominated by sensor processing or communications processing one might want data channel bandwidths and logic structure to be exactly like the target architecture. If, on the other hand, the target architecture is oriented toward an enhanced CPU repertoire, exact emulation of data channels and bandwidths is probably of secondary importance. The TSDF should have the flexibility to emulate data channels to the exactness required by the user. Known standard channels should be classified as a part of the TSDF data base. These should include byte, serial, word, and block transfer channels.

2.2.3.5 System Interconnect - The TSDF should have the capability of simulating, as a minimum, the system interconnect of the target architecture. The system interconnect thus simulated includes all of the channels and methods that transport information between the target architecture functional units. Emulation of the target architecture interconnect may be aspired to, but the difficulties in achieving the emulation must be recognized if one strives for performance equivalence. The simulated interconnect must reflect the characteristics of the desired interconnect: topology, bandwidth, point-to-point parallel bit transfers, link parallelsim, interconnect logical control, interconnect processing-like operations. Classes of interconnect that typify those parameters are:

- Bus system interconnects at the I/O channel level that are either serial, byte, word, or block organized, transporting information in rings, common connection, or star-substar topologies.
- Shared memory interconnects that commonalize the access to system memory among the functional system units, these interconnects may not have to be throughout the interconnect system.
- Bus interconnects that are normally internal to a single computer which allow transfer of information between processors, between processors and memories, between memories, etc.

2.2.3.6 Performance Measuring/Analysis (PMA) - The TSDF PMA should be capable of capturing predefined, event-associated data using the appropriate measurement technique (devices or software, for example). Performance-invariant data of the TSDF should be user-identifiable. Data collected during operations should be placed in a common repository for further reduction. Users should be given the option of collecting and reducing subsets of the predefined data. Explicit user selection of standard Performance Measurements (PMs) is desirable if standard PMs can be defined (related to HLHDL). Real-time PM data reduction and analysis is not required, but interactive reduction and analysis is highly desirable. (The user can request a "display" of reduced PM data immediately after a specified operation point - termination or a predefined system execution point.) Interactive operation with TSDF simulation is a problem.

Measurements capability should extend to the micro level but with tabulation capabilities only. Micro level operations are well regulated and timed, therefore many micro level operations do not need time-base measurement, but repetition measurements. Measurement parameters should be able to be equivalent to simulation parameters for extrapolating system performance.

2.2.3.7 TSDF Scenario Verification - The TSDF should have the capability of verifying the user-defined scenario of TSDF operations for obvious faults. This is analogous to the verification performed by compilers of user-generated high level languages. This capability should embrace the entire scenario through PMA and projections of target system performance.

2.2.3.8 TSDF Transformation Tools - The TSDF should include the following as functional capabilities for developing a system design using high level language:

2.2.3.8.1 Process Design Language - A language capable of providing a phased but detailed design for software components.

2.2.3.8.2 Hardware Design Language - A language capable of establishing the desired interface between hardware and software components (as opposed to the interface between two or more software components or two or more hardware components).

2.2.3.8.3 Requirements Specification Language (RSL) - A language capable of establishing the control structure (e.g., the responsibility for allocating resources) between all system components (hard or soft).

2.2.3.8.4 Provac Query Language - A language capable of permitting inquires to be made of the Process Visibility and Control (PROVAC) functional capability.

2.2.3.8.5 Process Visibility and Control - The logic which monitors all phases of system design for which tools exist and restructures the data for purposes of providing management information.

2.2.3.8.6 Translator Writing System - The logic which enabies the transformation of the lowest level of the PDL into a form (e.g., IEL) more easily converted to machine language.

2.2.3.8.7 Auto Code Generation - The logic which enables the IEL to be converted into machine language as defined by the HDL (possibly a subset of a more general purpose HDL).

2.2.3.8.8 Hardware Design Language (HDL) Translator - The logic which 1) translates the HDL into firmware capable of emulating the repertoire defined by the HDL, and 2) produces tables to drive the auto code generator.

2.2.3.8.9 Performance Requirements - A language capable of identifying the performance demands required of system components operating independently or in concert with other components.

2.2.3.8.10 Performance Parameters - A data structure (or language) capable of expressing performance measurements achieved through simulation, measured execution, or theoretical estimates.

2.2.3.8.11 Requirements Specification Language Translator - The logic which translates the RSL into directives (primitives) which drive dynamic system configuration logic (kernels) or programs (e.g., system generators).

2.2.3.8.12 Process Control Kernel - An extension to the conventional logic of a hardware processor enabling it to enforce the use of standard interfaces between components and permit software logic to dynamically change the configuration of components making up the system.

2.2.3.8.13 Performance Measurement - The logic which operates in conjunction with the kernel to provide performance measurement parameters derived during execution of the process structure (software portion of the system).

2.2.3.8.14 Integrated Hardware/Software Components - A methodology and associated functional capability which permits system configurations to be made with flexible interchange of hardware and software components. (The degree to which this can be automated is unknown. Answers to a number of questions will result from the use of the reconfigurable subsystem.)

In order to support the transformation tools, the TSDF should have a hosted data base management capability. Data base entries necessary to support the design tools and to provide visibility into the design process are:

2.2.3.8.15 Process Design Language (PDL) - All data submitted to the TSDF for purposes of establishing the logic sequence to be provided by an individual software component. At a minimum this must be a higher order language. Ideally, it includes structured logic sequences sometimes referred to as Pidgin English.

2.2.3.8.16 Hardware Design Logic - All data submitted to the RCSDF for purposes of defining the hardware architecture visible to software. As a minimum it must include a formal descritpion of the machine repertoire, e.g., Instruction Set Processor (ISP).[2]

2.2.3.8.17 Requirements Specification Language - A description of the required hardware and software components intended for the system. As a minimum it must show the control hierarchy.

2.2.3.8.18 Management Information (MI) - All data accumulated to enhance progress visibility relative to schedule milestones and financial expenditures.

2.2.3.8.19 Performance Requirements (PR) - All data indicating known performance requirements for individual components specified for the system.

[2] G. Bell and A. Newell, Computer Structures: Readings and Examples (1971).

2.2.3.8.20 Performance Parameters (PP) - All data showing measured or hypothetical performance characteristics of or between specified components of the system. Examples include execution times, delays due to required mutually exclusive resource requirements, and data flow rates.

2.2.3.8.21 Intermediate Exchange Language (IEL) - Language used to aid translation between PDL and executable machine code.

2.2.3.8.22 Generation Tables (GT) - Data structures produced by HDL translator enabling IEL to be translated into desirable machine code.

2.2.3.8.23 Object Code (OC) - Data capable of driving associated hardware processor element.

2.3 TSDF Functional Design Issues

To provide for the detailed synthesis of the TSDF system concepts for evaluation, the concepts, descriptions, and issues discussed in the previous sections must be formalized. To do this the design issues must be posed and then answered. Our procedure is first to develop the set of global system constraints on the TSDF and then to construct the TSDF function model and specialize it based upon issue constraints. Detailed models can then be constructed and traded, depending upon user requirements (wants) and desirable system features (system attributes).

2.3.1 Global Issues

Issue 1

How does the operational user measure TSDF capability, achievements, performance, and applicability to solve his system definition problems?

One of the major system issues involved with TSDF is how a user can decide if RCSDF is useful to him. It must be made clear to the user that TSDF has capabilities to solve certain classes of problems. Further, the user must see a benefit in using TSDF. Thus, it seems imperative that the user be able to see that TSDF will help him achieve a goal and that the performance and applicability of his design be measurable in terms of providing answers to guide system development.

Issue 2

Who are legitimate TSDF users?

It is not anticipated that a system like TSDF could (or should) solve every user's problem. The class of legitimate users can probably be narrowed by asking two questions:

1) Are the facilities of TSDF capable of supporting the user's application?
2) Is the user's problem significant enough to warrant use of TSDF?

Issue 3

What type of applications does the user desire to simulate?

To insure user acceptance of TSDF (i.e., to insure that TSDF is useful) a catalog of capabilities should be generated. In this catalog (or handbook) the equipments, configuration, software limitations, use, etc., of TSDF should be described so a user can evaluate the applicability of TSDF to his job or problem using a list of possible applications or an applicability checklist.

Issue 4

What does TSDF do for the user?

TSDF should provide the user with a tool to validate, detail, and tune his system design. Through the use of TSDF, the user should be able to hypothesize a design, its parameters, and its performance and then validate and tune the design to provide a tradeoff comparison of a series of designs. TSDF should provide a detailed output of the system performance, any bottlenecks, any potential bottlenecks, and any potential improvements possible in the system design.

Issue 5

What facilities does TSDF provide the user?

When attempting to utilize TSDF, the user will most probably see a software facility. The interface seen by the user should allow for hardware, system, and software specification, but TSDF should provide enough facilities to make it unnecessary for the user to deal directly with any particular nonsoftware facility.

Issue 6

Who maintains the facilities?

We assume that a centralized administration will be responsible for each TSDF system and will maintain the facility to ensure that the configuration not only is kept in working condition, but that the configuration is properly documented.

Issue 7

How does the user access and utilize the facilities to perform a job?

TSDF should be accessible as a resource via standard batch and/or time sharing facilities. The user will probably see the equivalent of a JCL (Job Control Language) and thus be able to sequence the capabilities of TSDF on his particular job.

Issue 8

When in the system design process does the user utilize TSDF?

TSDF should be the cornerstone of the design process. Once the initial system requirements have been generated, TSDF should be utilized to simulate approaches, refine requirements, and finalize the design.

Issue 9

What constraints does the TSDF place on either user or application?

Unfortunately it is expected that TSDF may not be applicable to all problems. In this context, we expect that the user groups to which TSDF is applicable can be categorized. Further, this description may be solely dependent on the performance issue, i.e., if the user needs high performance and can not get along with a system that is normalized in performance, then TSDF may not be applicable to that user's problem.

Issue 10

What facilities can the user add to TSDF? Do such expansions remain permanent?

A major issue - which we will not attempt to discuss at this point - is whether or not the user may add facilities or hardware to a TSDF. This includes whether such additions are temporary or permanent, how configuration control is maintained, and how a user is assured that the TSDF is performing with consistent results (i.e., there is no impact due to hardware added between runs of a specific user not involved with the hardware additions).

Issue 11

How are the results produced on TSDF delivered to the user, normalized, interpreted, and fed back into the design procedure?

Once a user has made a run (or set of runs) the results must be produced and displayed to the user. Other issues are involved, since the TSDF may not perform at the rate desired by the user, for example, the way performance results are normalized to produce the answers that would be observed in a real system. An essential issue in this regard is whether or not the user is allowed to interact dynamically (e.g., from a scope) with the TSDF. In an interactive mode the user could dynamically adjust the system parameters to produce optimum throughput.

2.3.2 TSDF Functional Model - A functional model of the TSDF is diagrammed in Figure 2-3. The sixteen basic functional components of the TSDF are:

1) System Definition Mapper - This function provides the interpretation of the user requirements necessary to develop configuration information to drive and configure

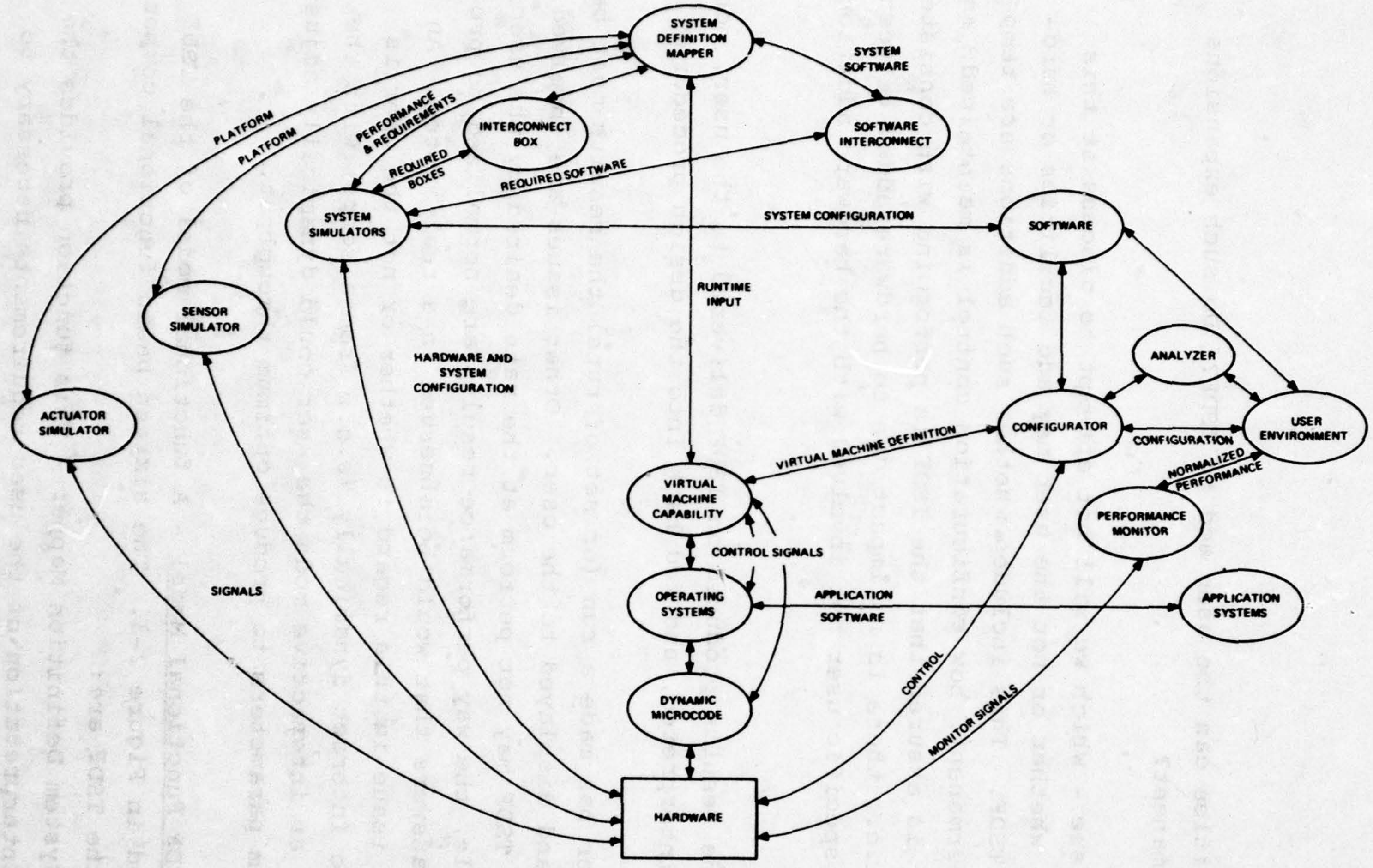


Figure 2-3. TSDF Functional Model

TSDF. Input to this function will describe major system design constraints. The output of this function will configure the system.

2) Interconnect Box - This is the hardware function that provides all of the hardware interconnection paths between TSDF hardware elements.

3) Software Interconnect - This function includes all software necessary to make TSDF function properly. It includes items such as loaders, compilers, and software. Further, any software certification or development features specific to an application must be integrated with this functional package.

4) System Simulators - This function contains any required simulation or emulation packages which are to be furnished to the TSDF hardware system. Multiple simulators (some software, firmware, or hardware) may be provided for a specific device to make hardware/software/firmware tradeoffs available to the user.

5) Sensor Simulators - This function provides the TSDF system with an environment delivery function which simulates sensor actions.

6) Actuator Simulators - Analogous to sensor simulators, this function accepts system inputs and drives simulated sensors.

7) Software - This function consists of any system software provided by TSDF and any user software used to drive TSDF configurations.

8) Configuration - This function takes the software/firmware/hardware user environment information and configures the detailed TSDF system to be used for an application.

9) User Environment - This function simulates the user's run

time environment. It is the result of all hardware/software/firmware manipulation and definition that has occurred to this point. Further, this is the main interface to the system that the user sees.

10) Analyzer - The analyzer functions to provide analysis and feedback on system performance. It accepts as input the normalized performance produced by the performance monitor and provides directions to both the configurator and user environment as to detailed performance bottlenecks, suggested configuration changes, etc.

11) Performance Monitor - The performance monitor function monitors the system performance and normalizes this performance based on configuration matching parameters (which are designed to compensate for basic speed differences that may be encountered between the system being simulated and TSDF).

12) Application Systems - This is the application software and/or special applications hardware which will be used to drive TSDF.

13) Virtual Machine Capability - To provide for efficient implementation of the general purpose system simulators a virtual machine capability may be required. This in effect will allow multiple users to run in protected environments or base hardware, thus providing significant speed advantages.

14) Operating System (OS) - This function is the application oriented OS provided for the user application on TSDF. This function may have to be user provided.

15) Dynamic Microcode - It is possible that the user will desire to tailor specific machine features, in which case this function should be provided.

16) Hardware - This consists of the fundamental hardware elements provided for TSDF.

For each of the TSDF functional capabilities, a list of implementation alternatives can be developed. For example, System Definition Manager could be implemented with software (however, this is quite unlikely at this point) or it could be performed by a person. Initially, this portion of the system probably can not be automated and will thus have to be performed by the TSDF user.

After considering the TSDF system design concepts and the questions and functional model posed in this section, we postulated an initial TSDF configuration. It is described in the next section.

2.4 A Viable TSDF Overview

The success of the initial RCSDF design study will depend upon an understanding, and ultimately the availability, of tools which enable an RCSDF to be easily used for system design. To further the understanding of the role a Reconfigurable System Design Facility (RCSDF) will have in design, a Total System Design Facility (TSDF) will be described as a background against which to describe the recommended RCSDF.

The functional capabilities required of a TSDF have been illustrated in an overview that depicts and discusses a reasonable set of design transformation examples which could potentially be supplied by such a facility. It is not intended to be an exhaustive set, nor does it necessarily contain the most feasible elements of an optimal set. However, the overview generally describes the direction in which system design must go in order to permit economical system deployment.

The described TSDF can only be partially achieved with the available technology for software and hardware, design and development. As such the TSDF serves as a focal point to help coordinate and integrate the latest and most feasible design/development techniques. In addition, several suggested capabilities have not been adequately researched. Thus the TSDF as described serves to promote meaningful investigations into potentially useful but as yet unproven methodologies.

2.4.1 Procedural Scenario - Figure 2-4 illustrates a TSDF consisting of five subsystems. In brief, the organized ideas for a deployable application system enter into system design by means of the thought processes of man (human subsystem). These thoughts are formalized and submitted to tools (hosting subsystem) which aid and simplify the decision-making process. Decisions for functional implementation by means of hardware or software must be made here. When the decision is made to implement with software, the development proceeds in the hosting subsystem. When a decision is made to implement with hardware, implementation specifications can be submitted by the hosting subsystem to normal industrial process control procedures (system integration subsystem). The integration subsystem has been depicted in dotted lines to indicate that this procedure is rapidly moving from a process requiring human intervention to one which can be totally automated, thus further reducing hardware costs.

When the human subsystem together with the hosting subsystem has determined the means for implementation, a critical phase of system design is entered. If the individual components of the system (hardware or software) can not function together to meet identified performance requirements, it makes no sense to enter

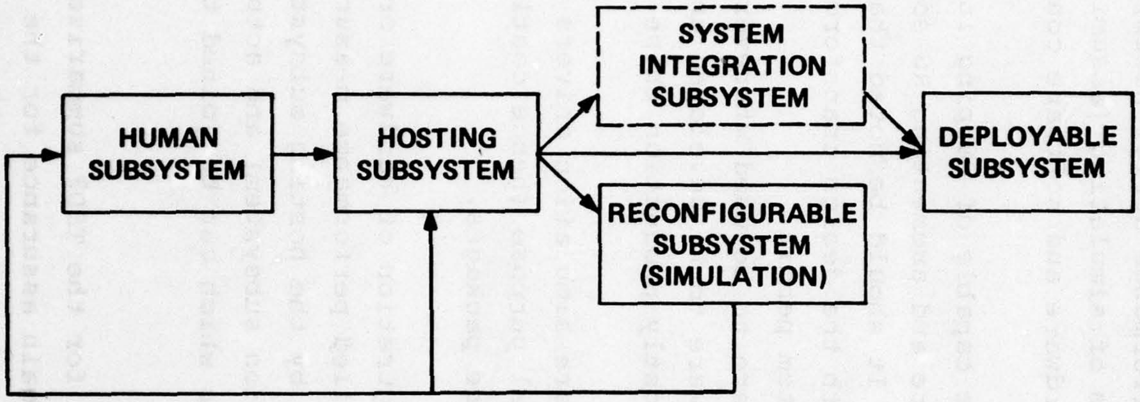

HUMAN SUBSYSTEM
HOSTING SUBSYSTEM
SYSTEM INTEGRATION SUBSYSTEM
RECONFIGURABLE SUBSYSTEM (SIMULATION)
DEPLOYABLE SUBSYSTEM

Figure 2-4. A TSDF

into the detailed design and production tooling phase for hardware (the most costly phase of hardware development). Thus a Reconfigurable Subsystem (RS) with controlled performance measurement (known performance deficiencies) becomes mandatory. The RS must be capable of simulating (assuming the characteristics of) the final hardware and software configuration.

The simulation must be capable of mapping intended deployable hardware to RS hardware and augmenting RS software with actual deployable software. It should be noted that the reconfigurable subsystem together with the design transformation tools available in the hosting subsystem permit:

1) existing software to be used wherever possible,
2) high risk software to be developed and tested before entering the costly production phase (of either hardware or software),
3) low cost software simulation drivers to replace straight forward, special purpose (hence costly) hardware peripherals and software packages.

When a feasible configuration of hardware or software can be verified under controlled performance measurements, specifications can be produced by the hosting subsystem for hardware development (integration subsystem) and actual software generated, providing the elements which can be joined together to form the deployable subsystem.

The overall objective for the TSDF summarized in the procedural scenario above is to gain assurance for the feasibility of the system design before going to the costly production phase and ultimately having to discard or correct the deficient system elements.

2.4.1.1 Human Subsystem - The term "design" refers to the conceptual or creative process required to achieve an identified capability, followed by the establishment of a plan for implementation. As such it can never be totally automated and will always involve a human factor. This is not to say that the TSDF can not be used during the initial design phase but rather that its function during this phase is support rather than decision making. It is best suited for the retention of information, feedback of information in requested format (e.g., what capabilities currently exist, what characteristics they display), and enforcement of a general plan for system implementation. The enforcement should be based upon successful design principles for data processing which might include:

Establish structuring standards
Commence evolutionary partitioning
Simulate and test potential system configurations
Utilize simulation results in evolutionary process
Implement high risk software components
Select optimal partitioning
Build remaining deployable components

The human subsystem portion of the TSDF becomes the means whereby the human factor communicates the potential design to those tools which can monitor progress, support design decisions, and automate portions of the development process useful for simulation. The primary means for communicating concepts between man and machine is through the use of syntactically structured languages. While languages vary, they fall primarily into two categories: those which resemble the natural written language and those which resemble various forms of mathematical notation. Since the components to be developed for an application system are a part of a digital and hence mathematically oriented computer system, both language forms are required, the first to communicate ideas between people and the second to lay out the sequence of operations required for a specific component implementation. Frequently the different language forms are referred to as levels in the same component design language.

Component design languages will vary. For software the language is referred to as the Process Design Language (PDL). Ideally the lowest or last level employed is referred to as a Higher Order Language (HOL) (e.g., JOVIAL, FORTRAN) and provides the means for expressing numeric and logical operations. Numerous outer levels have been suggested. Among the most promising appear to be those which combine natural languages with the logical block structures associated with structured programming. Figure 2-5 illustrates the use of this PDL level for the design of a real-time executive. It is important to note that the natural language can be replaced with lower language levels (i.e., HOLs). If this level of PDL is required, progress can be measured by making design corrections at this level when implementation problems are encountered.

Hardware Design Languages (HDLs) may require slightly different lower level language forms, e.g., Boolean algebra. For HDLs the outer levels have never been adequately examined. Nevertheless, meaningful interim levels have been defined, including the ISP notation developed for purposes of formalizing the description of a machine's repertoire. Figure 2-6 shows an adaptation of this language used to define a control algorithm (dispatch) which must be present in a reconfigurable system architecture. Figure 2-7 shows the ISP language used to define an instruction for a Univac 16-bit product line processor.

For a specific time frame, implementation constraints for TSDF capabilities must be identified. For example, whether or not current language technology will permit the outer level language

```
REGISTER MODULE ESR

IF PRIORITY IS VALID
. THEN
.  IF MODULE NAME NOT ALREADY IN MODULE LIST OR TASK LIST
. . THEN
. .  IF MODULE TABLE NOT FULL
. . . THEN
. . .  IF IPL INDICATOR IS SET AND A PSEUDO MESSAGE TASK EXISTS
. . . . THEN
. . . .  REPLACE PSEUDO MESSAGE TASK ENTRY WITH THIS MODULE IN
. . . .   MODULE LIST
. . . .  (NAME, PRIORITY, UN-SCHEDULED, UN-SUSPENDED, DEFAULT TASK ERROR
. . . .   MASK, NON-REGISTERED INDICATOR, NON-NRM INDICATOR)
. . . . ELSE
. . . .  PLACE THIS MODULE ENTRY INTO NEXT AVAILABLE SLOT IN MODULE
. . . .   LIST NAME, PRIORITY
. . . .  UN-SCHEDULED, UN-SUSPENDED, DEFAULT TASK ERROR
. . . .   MASK, N(N-REGISTERED INDICATOR, NON-NRM INDICATOR)
. . .  ENDIF
. . .  *ASSIGN STORAGE* (TO THIS MODULE)
. . .  IF ASSIGNMENT WAS SUCCESSFUL
. . . . THEN
. . . .  SET SUCCESSFUL STATUS
. . . .  SET REGISTERED INDICATOR FOR THIS MODULE
. . . . ELSE
. . . .  SET ADDRESS ERROR IN STATUS
. . . .  CLEAR MODULE ENTRY FROM MODULE LIST.
. . .  ENDIF
. . . ELSE
. . .  SET MODULE TABLE OVERFLOR INDICATOR
. .  ENDIF
. . ELSE
. .  IF NRM INDICATOR IS SET FOR THIS MODULE
. . . THEN
. . .  IF REQUESTED MODULE BOUNDARIES LIE WITHIN STORAGE ASSIGNED TO
. . . .   DUPLICATE MODULE
. . . . THEN
. . . .  *RELEASE STORAGE* (ASSIGNED TO DUPLICATE)
. . . .  *ASSIGN STORAGE* (REQUESTED BY THIS MODULE)
. . . .  SET SUCCESSFUL STATUS
. . . .  SET MODULE REGISTERED INDICATOR
. . . .  REPLACE DUPLICATE MESSAGE TASK ENTRY WITH THIS MODULE
. . . .   (NAME, PRIORITY, UN-SCHEDULED, UN-SUSPENDED, DEFAULT TASK
. . . .   ERROR MASK, NON-REGISTERED INDICATOR, NCN-NRM INDICATOR)
. . . .  SET MODULE REGISTERED INDICATOR AND NRM-IPL INDICATOR
. . . . ELSE
. . . .  SET INVALID ADDRESS STATUS
. . .  ENDIF
. . . ELSE
. . .  SET DUPLICATE NAME STATUS
. .  ENDIF
.  ENDIF
. ELSE
.  SET INVALID PRIORITY STATUS
ENDIF
RETURN

END
```

Figure 2-5. Example of PDL

```
DSPH/DISPATCH PROCESSING:  =
(ACTIVE PROCESS<SYSTEM> ←
PROCESS EXECUTION READY LIST   [SYSTEM]  , NEXT
PROCESS STATE   SYSTEM> ←  ENVIRONMENT
  <ACTIVE PROCESS <SYSTEM>>  :  NEXT
PROCESS ACTIVE  <SYSTEM> ←  1
DISPATCH ACTIVE   <SYSTEM> ← 0; CYCLE).
```

Figure 2-6. An Adaptation of ISP

```
OP CODE                         DESCRIPTION

(op=04 f=3)→                    ((byte-address;next
Byte Load &                     R[m]←R[m] + 1); next
Index by 1                      (m=0U¬R[m]<0>)→(R[a]<7:0>←M[Z]<15:8>);
                                (m≠0∩ R[m]<0>)→(R[a]<7:0>←M[Z]<7:0>);
                                R[a]<15:8>←0;
                                C←0;
                                OF←0;next
                                single←R[a];next
                                set-CC-single);
```

Figure 2-7. An AN/UYK-20 Instruction Description in ISP

definition for a HDL or, if an ISP level is defined, whether or not it can be translated into computer aided design controls which will permit processing chips to be built supporting the chosen repertoire must be discussed. The second portion of the RCSDF study identifies the anticipated time frames for which the technologies will exist and the implementation of the capabilities pursued.

A final capability required of the human subsystem is a means (or language) for management to interact with the hosting subsystem. This capability is further discussed as a part of the hosting subsystem.

2.4.1.2 Hosting Subsystem - The hosting subsystem supplies the logic which transforms a system design from the form submitted by the human subsystem into system component specifications, and, where achievable, process control for automated component development. The components may be either hard or soft. The hosting subsystem is seen as retaining a library of component designs complete with performance specifications. The design may exist in multiple stages of design transformations, e.g., higher level language description, machine object code, register transfer language description, and hardware logic diagrams/specifications.

There are at least six technologies involved in the transformations which take place in the hosting subsystem. Each is present in Figure 2-8, having been illustrated with examples of what is now available or being advocated by the research community. The technologies are discussed below.

2.4.1.2.1 Management Engineering - For the TSDF, management engineering is defined to be that logic which enables management to gain insight into the progress made in system design/development and permits management to enforce individual component

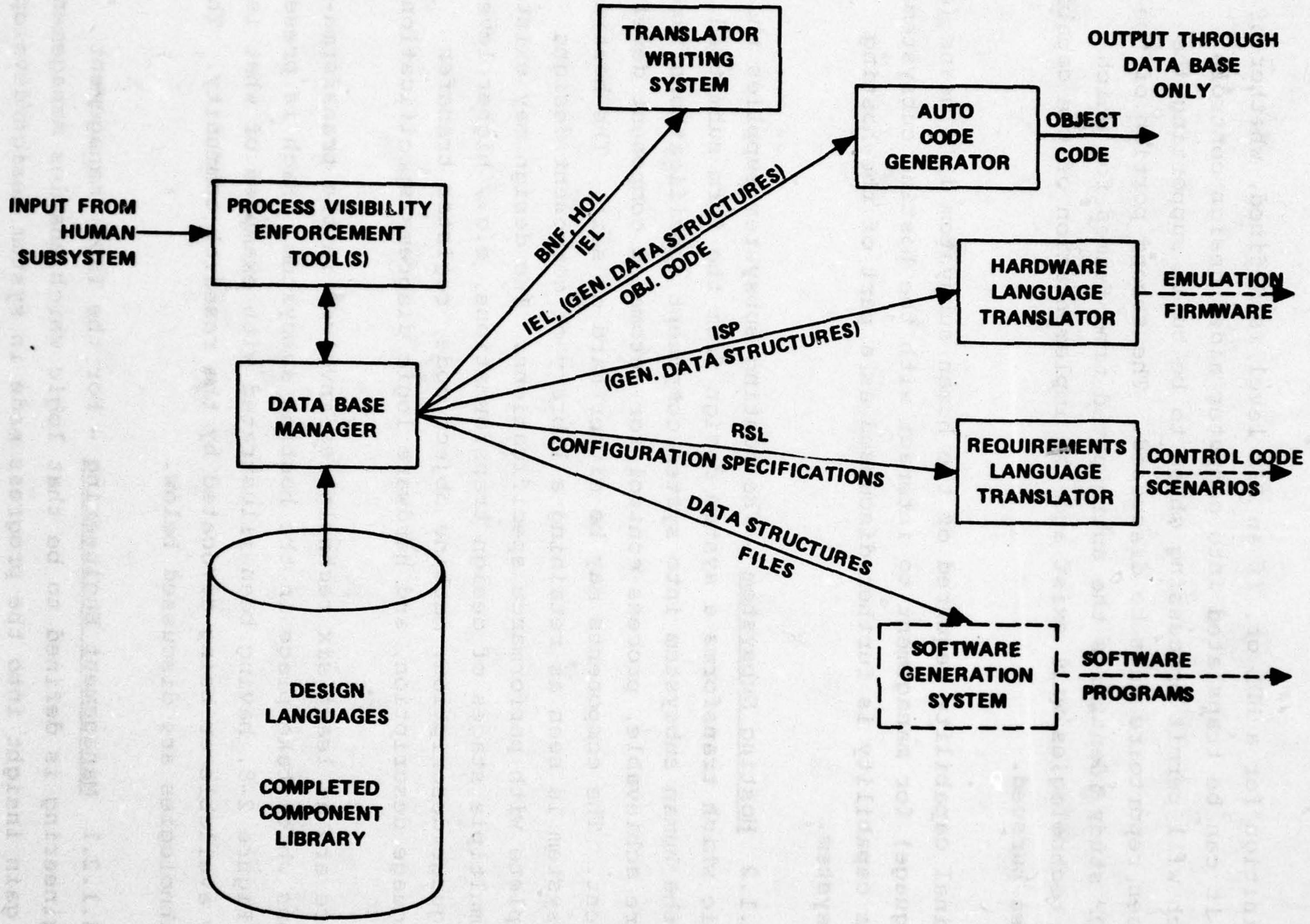

INPUT FROM HUMAN SUBSYSTEM
PROCESS VISIBILITY ENFORCEMENT TOOL(S)
DATA BASE MANAGER
DESIGN LANGUAGES
COMPLETED COMPONENT LIBRARY
TRANSLATOR WRITING SYSTEM
BNF, HOL
IEL
IEL, (GEN. DATA STRUCTURES)
OBJ. CODE
AUTO CODE GENERATOR
OBJECT CODE
OUTPUT THROUGH DATA BASE ONLY
ISP
(GEN. DATA STRUCTURES)
HARDWARE LANGUAGE TRANSLATOR
EMULATION
FIRMWARE
RSL
CONFIGURATION SPECIFICATIONS
REQUIREMENTS LANGUAGE TRANSLATOR
CONTROL CODE
SCENARIOS
DATA STRUCTURES
FILES
SOFTWARE GENERATION SYSTEM
SOFTWARE
PROGRAMS

Figure 2-8. Example of a Hosting Subsystem

quality assurance procedures. For example, each level of the process (software) design language can be retained in the data base for comparison with subsequent (i.e., more detailed) lower level language descriptions. If the contents of the language levels do not agree, this information can be recorded for management reports. This forces outer level descriptions to be updated and permits management to assess the impact on the remaining system components. Testing of an individual system component (e.g., program debug) is not considered to be complete until all levels of system design involving that component have been updated and all requirements unique to that component have been exercised and subsequently validated.

In Figure 2-8 the logic for management engineering is labeled "process visibility enforcement tool(s)." It should be noted that to take full advantage of management engineering technology, a tremendous improvement in the types and quality of design languages must be undertaken in the research community.

2.4.1.2.2 System Engineering - System engineering is defined to be the activities that support design, development, and ultimately integration of both hardware and software components such that they are consistent with specific application performance requirements. Inherent in this definition is the need for a rigorous decompositional technology based upon comprehensible theoretical models and management techniques which insure visibility and control.

A theoretical basis for decomposition compatible with the RCSDF study is the application of finite-state machine theory. This application would functionally decompose an application system into processes (components) each of whose potential interface with other processes of the system is delimited by the definition of a hardware state vector. By definition, a functional process is hard if it is the only process which may impact a machine's state vector.

Once the system is decomposed into functional hardware and/or software components, the performance requirements can be established for each component relative to its interface with other components. It is convenient at this point to organize the components in a hierarchical structure that illustrates which components control the use and allocation of system resources, including hardware devices, programs, and data.

The system engineering technology, like the other technologies handled by the hosting subsystem, requires a multilevel design language which has not been adequately defined. This language is herein referred to as the Requirements Specifications Language (RSL) and must be capable of describing system decomposition.

System engineering is represented in Figure 2-8 by the system requirements language translator. Two outputs are hypothesized. The first is an object subject code capable of invoking either resource control algorithms of the reconfigurable subsystem dynamically or procedure calls to statically drive a system generator (producing a software configuration). The second output is a scenario which controls the performance measurement logic and performance feedback available within the reconfigurable subsystem.

2.4.1.2.3 Software Engineering - Software engineering is defined as activity which supports the design and development of software components. The input to this technology from the human subsystem is called the Process Design Language (PDL). Examples of the contents of two levels of the PDL were discussed in the human subsystem section and the potential impact on the overall design procedure was discussed under management engineering. A required outer level of the PDL is a Software Requirements Specifications Language (SRSL) which describes the attributes unique to this component and against which system performance measurements can be taken.

The software engineering logic is represented in Figure 2-8 by two functional entities entitled Translator Writing System (TWS) and Auto Code Generation (ACG). While not including all of the functional requirements for software engineering, they do represent two important design transformations necessary for total system design. Translator writing systems have been achieved using existing software technology which permits a variety of inner level PDLs (i.e., HOLs, e.g., JOVIAL and FORTRAN) to be translated into a common Intermediate Exchange Language (IEL). As shown in Figure 2-8, the actual translation process is controlled by describing the language syntax using yet another language (e.g., Backus Nour Form).

Once the process (component) design has been specified and translated into an IEL it is technically feasible to translate the IEL into target machine repertoire sequences. For total system design, however, the repertoire chosen is a part of the design process. As such, the last translation process (called Automated Code Generator) would be driven from the hardware language translator and is discussed as a part of hardware engineering.

2.4.1.2.4 <u>Hardware Engineering</u> - Hardware engineering is defined to be that activity which supports the design and development of hardware components. The input to this technology from the human subsystem is called the Hardware Design Language (HDL). A discussion of potential language levels (e.g., ISP) was included in a discussion of the human subsystem. A required outer level of the HDL is a Hardware Requirements Specifications Language (HRSL) which describes the intended characteristics of the hardware component being defined, e.g., memory access time, machine cycle time, and tolerable remote access delay times. The requirements specifications can be utilized as factors to offset the controlled performance measurements returned to the human or hosting subsystems from the reconfigurable subsystem.

The logic required to support the hardware engineering technology is represented in Figure 2-8 by the entity entitled hardware language translator. The hardware language translator is depicted as producing at least two outputs. The first is Generation Data Structures (GDSs) which supply the ACG with a description of the repertoire defined for a "target" machine. From the GDSs it is hypothesized that the ACG is able to translate the IEL into the repertoire as defined by a level of the HDL similar to ISP.

The second output supplies (perhaps as firmware) the repertoire selected by the design to the RS, thus enabling it to simulate the intended target machine. An alternate path would allow the repertoire specifications (including requirements specifications) to be passed on to logic designers, enabling specialized processors to be developed for the deployable subsystem.

It should be noted that we have just described a means for automating the first level of interface between hardware and software. A second level, that of component sequence control, is superficially discussed in the reconfigurable subsystem.

2.4.1.2.5 <u>Data Base Management</u> - For the TSDF, data base management is defined as that logic which exists for storing, retrieving, updating, deleting, and otherwise manipulating data which represents the current state of system design. The concept of large data bases which can be easily accessed by systems such as the human subsystem and hosting subsystems are well understood and documented. However, the techniques for implementation are continually undergoing evaluation. It is recommended that for the hosting subsystem, the method of implementation undergo extensive evaluation. This is particularly true if the hosting subsystem functions are ever to be mapped onto the reconfigurable subsystem.

2.4.1.3 <u>Reconfigurable Subsystem (RS)</u> - The RS proposed for the TSDF was conceived to permit desired application functions to be simulated using a variety of configurations of both hardware and software. The simulation permits performance measurements to be taken in order to access system bottlenecks and establish the feasibility of the overall system design. If bottlenecks are discovered which would tend to indicate performance failure in the deployable subsystem, corrective action can be initiated involving such measures as reconfiguration, hardware/software tradeoffs, parallel or multiprocessing, centralized data access, or hardware element usage. In addition, by establishing a facility which supports the measures discussed above, system capabilities can be developed which will promote the expansion of off-the-shelf software. The availability of these capabilities in the RS will permit software to be developed before hardware development is initiated and high risk software elements to be fully developed and tested. The result in both cases is a reduction in initial and/or life cycle system cost.

To achieve the desired characteristics, there are at least three uniquely identifiable capabilities which must be provided by the RS. They are listed below.

2.4.1.3.1 Tailorable Emulation Capability - The ability to take on firmware or nanocode to permit changes to the machine repertoire emulated by each processing element in the subsystem.

2.4.1.3.2 Performance Measurement Capability - The capability for monitoring and gathering performance statistics such as frequency of memory references, shared data domains required by individual software components, delays caused by mutually exclusive resources accesses, and frequency of component execution. To achieve this type of performance, measurement will require a hierarchical control of resource allocation which can be enforced by hardware but monitored by software as directed by performance requirement scenarios. A second potential requirement is dedicated hardware elements for data reduction and feedback to either the human subsystem or hosting subsystem.

2.4.1.3.3 Resource Control Algorithms - The desired characteristics for the RS require an enforced methodology for the control of system resources. Research projects have shown that a methodology can be enforced using control algorithms which standardize component interfaces in the system. Implementation of the algorithms, like the implementation of a floating point add instruction, can take numerous forms, including both hardware and software logic but the overall result of their invocation must be predictable or constant for the system architecture.

The set of resource control algorithms required for the reconfigurable architecture must be complete. This permits software modules to be added to the system configuration without modifying any portion of the software system already working.

For this discussion, control algorithms are said to be complete if, given a hardware configuration of processing elements, an existing stand-alone system of software can be reproduced to execute concurrently on the hardware without modification to the software and using only the extension to the machine repertoire furnished by the control algorithms. The reproduction must be possible either statically or dynamically.

To clarify, recall the function of a system generator program. Typically, a system generator accepts outputs from language processors (compilers) and, based upon directives supplied by system programmers, binds the software in a form capable of replacing the existing software in the system. Should a system have control algorithms demonstrating the property of completeness, a system generator would be capable of causing the original software plus a reproduction to execute concurrently without software modification and using only the constant function(s) supplied by the algorithms. Obviously, if the hardware remains constant, the throughput of one of the stand-alone systems might be reduced even though the control algorithms are shown to be complete. This feature is desirable, however, since theoretically off-the-shelf software functions can be added to the system, e.g., a missile tracking and display function added without change to the existing software. If the system of control demonstrating completeness is included as a part of the reconfigured architecture, the throughput of the system can be adjusted with the addition of hardware elements.

To aid in identifying a set of resource control algorithms which demonstrate the property of completeness, Univac has chosen the methodology called process control. The methodology of process control identifies a hardware system as dynamically decomposable into virtual machines, each of which can be defined as a concise summary of the state of the real machine for a bounded set of logic, i.e., a finite set of descriptors defining the effect that past inputs will have on the real machine's response to the next input. A process is defined as the behavior of a virtual machine executing a "program", i.e., the program, the machine state which the program assumes during a specific invocation, and the data which is unique to this invocation. Simply stated, each programmer or uniquely identifiable use of the system sees the program or module created or invoked as excuting on its own machine.

2.4.1.4 Deployable Subsystem - The deployable subsystem is the result of the design process and hence the TSDF facility. It must be reliable and maintainable and it must meet the needs of the application including high performance demands. It must be easy to use in order to reduce training costs and must provide flexibility for growth resulting from inevitable and evolutionary system demands. Most important, all of these characteristics must be achieved in a manner that will enable the life-cycle cost of the system to be dramatically reduced.

To achieve these characteristics the deployable system must be configured using both hardware and software functional components: hardware to increase performance, and software (or firmware) to permit flexibility without the retooling expense now associated with hardware development. The TSDF enables designing to proceed in a manner which provides management visibility at critical points. The visibility allows corrections to be made on detected design errors which have in the past gone undetected and later resulted in costly implementation.

2.5 TSDF Advantages and Risk Assessments

2.5.1 TSDF Advantages - The advantages of the TSDC methodology and by implication the TSDF has advantages that fall into three categories: software, hardware, and the system in general. They will be listed and explained in the sections below.

2.5.1.1 Software

2.5.1.1.1 Ordered Program Structure - Programs (and their hardware alternatives) can be developed, implemented, and tested to determine the execution behavior of their control and logic paths in as simple or complex an environment as is desired at the different system development times. (A system design does not evolve as a unit step function.) Further, these execution properties can be performance extrapolated to determine implementation requirements prior to design committal.

2.5.1.1.2 Transferability of Programs - The emphasis on HOL ensures a better transferability of programs between machines. Better transferability implies that developed program modules have a much better chance of being used in more than one system development, diminishing the overall cost of the module, and hence software, to the Air Force.

2.5.1.1.3 Program Correctness - A major advantage provided by TSDF is that a program can be viewed in two ways. First, it produces the expected transformation given prescribed inputs. Second, it behaves totally as expected in its operating environment - it interfaces properly with other programs with which it is merged, it operates concurrently without logical problems, and its performance level is within prescribed system margins. By hosting a complete system and exercising that system responsively to the expected environment, program correctness can be verified and the objective of zero fielded faults approached.

2.5.1.1.4 Program Generation - HOL programs developed on TSDF as a part of the system design and development process automatically provide programs for use in deployable systems. Transferability of the HOL programs to the implemented target architecture from the TSDF could easily be made.

2.5.1.1.5 Program Size/Execution Profiles - The TSDF is advantageous in optimizing software size/execution profiles by providing the means for examining alternative designs and determining overall system level performance differences. Emphasizing optimization analyses through TSDF methodologies will definitely aid in improving performance and limiting cost.

2.5.1.1.6 Support Software - The TSDF would be advantageous to the development of some support software elements for a TSDF-designed system. Following the principle that the TSDF faithfully emulates the target system and employs hosted software, it follows that support software, which usually doesn't have real-time requirements, can be developed using the emulation capability. Advantages are a longer software lead time, allowing necessary support elements to be present at the time the system is integrated.

2.5.1.2 Hardware

2.5.1.2.1 Flexibility - By providing designers with a TSDF capability to examine design and architecture alternatives, the same logical system structure can be used to achieve flexible system solutions to meet various cost-performance levels. Using the TSDF, a designer can investigate configurations of various combinations of his hardware and software resource modules at a very low design cost. He can then afford to choose the best configuration of his resources without doing the costly rewiring and recompiling. After a system has been built, the effects of adding a resource module to the system can be easily assessed using the TSDF. Hence the designer can avoid unnecessary hardware costs.

2.5.1.2.2 <u>Software/Firmware/Hardware Tradeoffs</u> - Because of the modularity of the system, it will be much cheaper and easier to investigate the software/firmware/hardware tradeoff problem. A hardware or firmware piece can first be emulated in the TSDF. By comparing the results with the original software version, the final decision can be justified more objectively. The emulation work can actually alleviate some of the designing and testing burdens later on, should the designer finally decide on building the hardware/firmware.

2.5.1.2.3 <u>Hardware Complexity Estimates</u> - The progression of development on the TSDF will naturally lead to the definition and specification of a hardware design, possibly of hardware components that do not exist. From this development effort the hardware complexity of the eventual system can be more accurately approximated and examined. Many deficiencies develop due to unanticipated development of hardware complexity in order to meet re-defined system functions, growth, or performance.

2.5.1.2.4 <u>Cost Bounding</u> - The firm definitions of the system architecture confirmed by confident performance evaluation can be expected to limit the cost of the system or cost-approximate the eventual system design.

2.5.1.2.5 <u>Risk Minimization</u> - The overall effect of TSDF is to provide preliminary design proof for a system design. The successful accomplishment of this results in the minimization of risks in the development of a system design, a significant advantage to a system development program. To the extent that risks can be minimized, or eliminated, the program exposure to time and cost overrun is reduced. The TSDF is seen to provide many advantages in this particular area.

2.5.1.2.6 Maintainability - The delineation of hardware, if carried sufficiently far in TSDF, can provide insight into maintainability requirements which, if they become extensive, can reflect themselves into high system operating costs.

2.5.1.3 System

2.5.1.3.1 Testing - Programs and systems designs can be more thoroughly tested and their logical operations validated with the aim of reducing fielded faults to zero. To the extent that fielded faults approximate zero, redesign costs will be minimized.

2.5.1.3.2 Reliability - The ability to exercise and evaluate programs on an emulated architecture prior to specification will help to increase the reliability of a system design by isolating software design errors, by allowing system degradation alternatives to be examined, and by allowing the system to be driven in an environment much like the application environment. In addition, since through TSDF evaluation the system design is firmly developed, the system hardware reliability can be more accurately approximated and parametrically analyzed so the system components can be chosen to achieve the greatest reliability for a specific cost.

2.5.1.3.3 Modularity - A key to more cost-effective systems designs seems to be modularity, both hard and soft, that allows systems to be expanded and modified without an avalanche of design changes. Modularity, for example, stresses isolation of design decisions to independent modules, thus allowing correction without affecting modules. Systems can be altered more easily if a module design is developed on independent module principles. A TSDF allows function movements between alternate hardware or alternate software modules or between hardware and software modules. Through modular design a module interface standard will evolve that supports independence, a characteristic lacking in current designs.

2.5.1.3.4 Modifications - The use of the TSDF to develop a system using HOLs and the proposed evaluation technique will allow modifications to be made more easily and less expensively since the system behavior has been observed and the system is better documented. Further, with data basing techniques the system can easily be recreated, modified, and tested in considerably shorter time frames.

2.5.1.3.5 Local Optimization - Although modularity stresses as much uniform handling as possible, the user is still allowed to investigate any local optimization within a module according to his needs. This is achieved by allowing users to generate their own emulated microcode sections, if they wish. A similar allowance can be done to the HOL compiler. In this way intramodule optimization can be ensured, if desired.

2.5.1.3.6 Partitioning - The alternative architecture evaluations possible on a TSDF will allow a user to develop significant insight into the logical and physical partitioning of his problem on the system. This insight can allow natural fault tolerance to be incorporated into the design to provide positive graceful degradation modes. Given further fault tolerance, proof of performance in a degraded mode can be developed.

2.5.1.3.7 Growth Approximation - Through development of system algorithms and their performance evaluation on different TSDF hosted architectures, system growth possibilities (or margins) can be more easily determined. This is an important determination since systems are seldom static. System functions, during system life, are modified or added resulting in system design changes. Growth margins have never been easily and precisely determined, either from an added hardware, added software, or performance standpoint.

2.5.1.3.8 Documentation - The use of HOL, EDL, and in general, more English-like algorithmic languages at the onset of a design, and the enforced use of HOLs in modifications during the design process cannot but help ease documentation of systems design simply because the languages are self-documenting. Modifications or the resolution of errors are often expensive because programs are poorly documented and written in assembly languages. Software is a creative art, with programs representing creative solutions to processing problems: one person's creativity may be another's nemesis.

2.5.2 Risk Assessments of TSDF Functional Capabilities - An initial evaluation on the technical risk levels for each of the TSDF functional capabilities described earlier in paragraphs 2.2, 2.3, and 2.4 is presented in this section, under the assumption the TSDF implementation will be completed by 1985. For each functional capability, this report will assign a risk level and the confidence level on this initial assessment.

The risk levels are listed below, along with an explanation of each level.

Level 1: Exceptionally Low

The technical risk is minimal. The risk subject will probably be produced as a natural technical development well before the anticipated need and several versions of the subject will probably be available for usage. It would generally require no industry encouragement to realize.

Level 2: Low

The technical risk is slightly greater than level 1 but presents no serious technical challenges should development be desired. Practical realization of the subject would probably occur near the anticipated need and would be supported by several experimental versions. It would require industry encouragement to accomplish.

Level 3: Medium

Technical risk in this area is moderate and represents a technical challenge to recognized practices. Subject realization in the needed form probably would not occur at the time of anticipated need unless encouraged and funded.

Level 4: High

This represents a serious challenge to the current technology. Subject realization will not occur unless heavily encouraged and funded. Research and development are required in order to make this function available at the required time.

Level 5: Exceptionally High

Technical developments in this category would require filling gaps in technical knowledge to be realized in the specified time frame. They would require many years of heavily funded research and development to realize. It would be necessary to rely on funding by both government and industry to realize this development.

For risk levels assigned as 4 (high) or 5 (exceptionally high), a post-1985 time frame at a reduced risk level will be projected, where possible.

If the risk level is dependent on alternative implementation of the subject, it will be described as a range, i.e., low to medium with subject alternatives described at either end of the range.

In the case by case assessment below, a confidence level of 1 means that not enough information is gathered about the subject, and the risk level assigned is an educated guess. A confidence level of 2 implies that we have more knowledge on the subject and are fairly confident about the risk level assessment. If a confidence level of 3 is indicated, it means that we are quite certain of the risk level assignment.

2.5.2.1 Simultaneous Hosting (on Reconfigurable Subsystem)

2.5.2.1.1 Single User

Risk Level: 2-4   Confidence Level: 2

This implies that the TSDF hosts only one emulation-user at any particular moment. This single user hosting is currently being performed in several places with existing general purpose emulation equipment. Risk level 2 represents an emulation of a known unit machine operating in a single synthetic measurement environment (instruction mix benchmark, for example). Risk level 4 represents an emulation of perhaps a parallel machine operating in response to an external stimulus with a reasonably competent measurement capability. The risk level may drop from 4 to 3 by 1990.

2.5.2.1.2 Multiple User

Risk Level: 5 Confidence Level: 2

Multiple user emulation is interpreted to mean the capability to multiprogram more than one user with the implication that the user-emulated machine varies from user to user. Multiprogramming on a single conventional machine is at this time still not without problem, and the multiple system emulation would only create a heavier burden. Moreover, it is doubtful if hardware technology will be advanced enough to handle this multiprogramming effort by 1985. The risk level will probably be lowered to 3 in 1995.

2.5.2.2 User System Timing

2.5.2.2.1 Real Time Operations

Risk Level: 2-5 Confidence Level: 2

Real time operations mean the ability of the emulated system to equal or better the performance of the user-designed system, so that all user system operations can be done in real time. Risk level 2 represents emulation of low performance existing machines. Risk level 5 represents the emulation of high performance machines. No reasonable time frame projection can be given to reduce the level 5 risk.

2.5.2.2.2 Simulated Timing

Risk Level: 3 Confidence Level: 3

It seems readily possible to simulate timing at the instruction level by incrementing a register. It would be necessary to develop this as a part of microcode development but would be intrinsic to the TSDF.

2.5.2.3 System Environment (Deployable System)

2.5.2.3.1 Synthetic

Risk Level: 2-3 Confidence Level: 2

Risk level 2 represents essentially state-of-the-art today, synthetic benchmarking, in which a prescribed code segment is executed n times and measured using internal system clock sources (real-time clock, for example). Risk level 3 represents the use of devices peripheral to the emulation system to interact with the emulation as an external stimulus.

2.5.2.3.2 Real

Risk Level: 3-4 Confidence Level: 2

Risk level 3 represents a simple, but realistic real environment with duplication of the system peripherals, and the ability to duplicate the behavior of the environment into which the emulation system is merged. Risk level 4 represents what would be a large system environment when complex stimulus is required and the emulating system is capable of operating in real time. The nearest known effort to this capability is the work being performed at the Ballistic Missile Defense Advanced Technology Center in Huntsville. Risk level 4 capability may be achievable at a risk level 3 during 1990 depending on the extent of software development tools.

2.5.2.4 Virtual Environment

2.5.2.4.1 TSDF

Risk Level: 3 Confidence Level: 2

The virtual environment on TSDF allows the user to assume any kind of machine. A risk level of 3 is assessed because of some uncertainty on the different kinds of user-designed systems that are allowed.

2.5.2.4.2 User-Designed System

Risk Level: 2-3 Confidence Level: 2

This implies that a user is allowed to host a virtual environment as part of an emulation. Several virtual resource techniques exist today and are in use, so the risk level is not high.

2.5.2.5 Architecture Emulation

2.5.2.5.1 Existing

Risk Level: 2-4   Confidence Level: 3

Existing architecture is defined as pre-1985 architecture. Most of these will have adequate descriptions upon which to base emulation. It will be a readily attainable capability for normal machines that has already been demonstrated in a number of places. However, the risk level will rise with deviance from normal machine architectures.

2.5.2.5.2 New

Risk Level: 3-5   Confidence Level: 2

This is defined as the emulation of post-1985 architecture. The risk level can be as low as 3 for new machines developed in traditional manner; it can be as high as 5 for revolutionary machine designs. No reasonable time frame projection can be given to reduce the level 5 risk.

2.5.2.5.3 Modification

Risk Level: 3   Confidence Level: 3

This is defined as the ability to modify existing components or their arrangement without reconstructing the complete system emulation. The problem lies in constructing an adequate description of components and the relationship between them. Emphasizing system modularization will tend to keep the risk level as low as 3.

2.5.2.6 Dynamic Architecture Alterations

2.5.2.6.1 Hardware Replacement

Risk Level: 3   Confidence Level: 2

This is defined as the dynamic selection of a present TSDF hardware capability by the user.

2.5.2.6.2 Hardware Addition

Risk Level: 3 Confidence Level: 2

This is defined as a piece of hardware added to help the user to achieve a more realistic result. A risk level of 3 is assessed on the assumption that an adequate interfacing method has been established.

2.5.2.6.3 Software/Hardware Interchange

Risk Level: 3 Confidence Level: 2

The TSDF should be capable of mapping software components to hardware processing elements without redesign. This means that a certain function can be performed via either software or hardware, with no major redesign. This requires a standard interfacing methodology to control various components.

2.5.2.7 Multiple-Machine System Emulation

Risk Level: 3-5 Confidence Level: 2

Risk level 3 represents the emulation of a homogeneous set of machines of moderate dimensions, complexity, and simple performance. Risk level 5 represents the emulation of heterogeneous machines, possible based on distinctly different computing principles, and is not seen to be reducible to level 4 or 3 before 1995.

2.5.2.8 Memory Configuration Emulation

2.5.2.8.1 Hierarchical

Risk Level: 3 Confidence Level: 2

Techniques already exist now for hierarchical storage; the problems appear to be translating these techniques to local configurations.

2.5.2.8.2 Shared

2.5.2.8.2.1 Centralized Sharing

Risk Level: 3 Confidence Level: 2

Examples exist and are being used quite widely.

2.5.2.8.2.2 Network Sharing (Software Intervention)

Risk Level: 4 Confidence Level: 2

This implies the use of software (for example, protocols) to help move data packets to desired locations. It is not likely that the risk level will be reduced to 3 until 1990.

2.5.2.8.2.3 Distributed Sharing (Hardware Defined)

Risk Level: 3 Confidence Level: 2

This implies a closely coupled system, in which information transfers are controlled more by hardware than by software.

2.5.2.9 Data Path Emulation

2.5.2.9.1 Direct Access

Risk Level: 3 Confidence Level: 2

The TSDF should have the capability of simulating the direct access system interconnect of the user-designed architecture. The risk level depends on the degree of exactness or complexity of the data paths.

2.5.2.9.2 I/O

Risk Level: 3-4 Confidence Level: 2

The risk level of 3 is assigned for standardized data channels. The risk level of 4 refers to nonstandard connection methods. Risk level could be reduced to 3 by suitably restructing the topologies.

2.5.2.10 Hosted System Preservation (Reconfigurable Subsystem)

Risk Level: 3-5 Confidence Level: 2

The TSDC should have the capability of preserving all particulars of a hosted system design including a capability for restarting the system from a termination point, should that be desired. If the termination point is arbitrary, then the risk level is 5 (very high). If enough checkpoints are established in the system, and the restriction relaxed to restarting from the checkpoints only, then the risk level will be reduced to 3.

2.5.2.11 Scenario Verification (Hosting Subsystem)

2.5.2.11.1 Requirement Specification Validation

Risk Level: 4 Confidence Level: 1

The hosting system should have the capability to systematically verify the validity of the user's system. No adequate requirement specification language currently exists. Development of a formalized requirements specification language will require time and would be a level 3 risk by 1995.

2.5.2.11.2 Execution Scenario Generation/Translation

Risk Level: 3 Confidence Level: 2

This implies the translation of a user-defined scenario into emulation driving functions. The risk level is medium, assuming we have high level language representation and approximately 85 to 90% accuracy for verifications.

2.5.2.11.3 Report Generation

Risk Level: 2 Confidence Level: 3

Many report generation techniques exist, so the risk level is low.

2.5.2.12 Performance Monitoring (Reconfigurable Subsystem)

2.5.2.12.1 Configuration Requirement Specification Translation

Risk Level: 4 Confidence Level: 1

For performance monitoring, we need information on user system configuration. This translation intends to produce a form of specification acceptable for monitoring. The uncertainty in an acceptable specification language is the main reason for the high risk level. It will be reduced to 3 by 1990.

2.5.2.12.2 Performance Scenario and Requirement Specification

Risk Level: 3 Confidence Level: 2

In the scenario, the user describes what performance measures are desired. He also specifies his performance requirements, i.e., acceptable bounds for certain performance measures. A risk level of 3 is given, on the assumption that the specifications can be represented in terms of high level languages.

2.5.2.12.3 Simulation Performance Measurements

Risk Level: 3 Confidence Level: 3

Performance measurement and evaluation techniques have been developed for many years. However, until more research has been done to incorporate performance measurements into emulation techniques, the risk level will be 3.

2.5.2.12.4 Adjusted Simulation Performance Report Generation

Risk Level: 3 Confidence Level: 2

This is the capability to extrapolate measured performance to the target architecture, i.e., being able to substitute expected performance parameters for measured performance parameters.

2.5.2.13 System Definition Languages

2.5.2.13.1 Process Design Languages (HOLs)

Risk Level: 2 Confidence Level: 2

Many such languages exist and many more will be developed. Thus the risk level is rather low.

2.5.2.13.2 Emulation Design Languages

Risk Level: 4 Confidence Level: 2

Most known hardware languages are on the logic level. Others do not have enough features to accommodate some of the new architecture features. It should be reduced to level 3 in 1990.

2.5.2.13.3 Requirements Specification Language

Risk Level: 4-5 Confidence Level: 1

The requirements specification language represents the combined problems of specifying 1) system function and performance requirements, 2) component function and performance requirements, and 3) control structure and resource requirements. Only 2) has been achieved with any degree of acceptance but even there, problems still exist.

2.5.2.14 Adjusted Design/Implementation Tools

2.5.2.14.1 Automated Software Design Translation

Risk Level: 4 Confidence Level: 1

This is interpreted as correlating the process design language programs with a requirements specifications language description. The high risk level is due to the uncertainty in requirements specification languages.

2.5.2.14.2 Automated Design/Machine Code Transformation

Risk Level: 3 Confidence Level: 2

This is closely related to compilation of high level languages.

Because of the advances in compiler design, the risk is low. However, to make the emulating facility a generalized one, the risk level is increased to 3.

2.5.2.14.3 Automated Machine Code Implementation

Risk Level: 3 Confidence Level: 2

This is the implementation of the user designed machine code into RCSDF firmware or software. The risk is low, but, to make the emulations facility a generalized one, the risk level is increased to 3.

2.5.2.14.4 Automated Hardware Logic Implementation

Risk Level: 5 Confidence Level: 1

This is the automatic generation of hardware logic design details from the emulation design language. Too little is known on this subject to predict with any certainty. The risk level might be reduced to 3 by 1995.

2.5.2.15 Requirement Specification Language Translation

2.5.2.15.1 Configuration Specification

Risk Level: 4 Confidence Level: 1

Here it is assumed that in addition to those requirement specifications discussed in 2.5.2.13.3, the user also has to specify which components are to be implemented in hardware and which in software. The translation of such specifications into some process design languages is considered high risk. The risk level should be reduced to level 3 in 1990.

2.5.2.15.2 Implementation Selection

Risk Level: 5 Confidence Level: 1

The translation will be even tougher if the hosting subsystem has to make an intelligent selection of implementations from a specification library to build up the user designed system. No reasonable time frame can be given to reduce the high risk.

2.6 Additional TSDF Issues and Recommendations

The initial investigation of TSDF functional capabilities leaves questions concerning a recommended course of action for the development program. These issues are related to one another but are described separately.

2.6.1 <u>One TSDF vs. Many</u> - Differences between system architectures can be great: networks vs. a single machine, for example. It is not clear that a single general purpose TSDF is adequate for development of a wide range of architectures. Associated with this is the concern about the accuracy associated with this emulation capability, especially when procurement may be based on projected performance. Perhaps more than one TSDF is needed to ensure the emulation accuracy needed by the user: it is felt that the greater the amount of performance is projected, the greater the chance that the implemented performance will not be as needed.

In the current state of the art in emulation/simulation, development apparently proceeds on the actual hardware (or functional development models of such) that will be used in the final system when performance is the major objective.[3] This is not expected to change and points to many TSDFs. The prospect of a multiple TSDF approach is also strengthened by the peculiarity and variability of the system periphery (peripheral devices and their usage). The recommended approach is to evolve from a parent TSDF

[3] BMD Advanced Technology Center, <u>op</u>. <u>cit</u>.

employing virtual memory concepts to sibling TSDFs whose characteristics resemble the parent, but whose actual performance (through designed and built components) is near the target architecture. Logically, parent software would perform on the sibling. Configuration documentation is an important implication in the multiple-TSDF approach.

2.6.2 Emulation as an Aspiration - Traditionally, emulation has been applied to mean performance-equivalent execution of object code (machine instructions). One machine, however, could emulate another at a lower level of performance. In the TSDF, what is emulation and what is simulation is unclear because of the different levels that are present. At higher system levels emulation is present in the TSDC, but in practice the TSDF may have to simulate a large part of the architecture for economic reasons. The best results would seem to be obtained if the TSDF were a system emulator rather than a system simulator. One concern here is the system interconnect problem and the difficulties of emulating the operations of a complex interconnect.

Current emulation technology centers on a one-to-one correspondence between the emulated and emulator even though general purpose emulation is professed. Recognizing that system emulation (as in the case of a distributed system architecture) is not well advanced, general purpose system emulation is even less advanced. Better general purpose emulators will be available in the future, perhaps sometime in the 1980's, for single machines but not for heterogeneous systems of machines. It is recommended that the present TSDF concentrate on a single machine general purpose emulator with the capability for utilizing other processing logic as needed. System emulation should be a function of sibling TSDFs, which would use hardware more like that of the target architectures.

2.6.3 Hosted Capabilities - Hosting of generation capabilities has been described as a part of the TSDF. It has been assumed that this would be accomplished on facilities separate from TSDF hardware. Could the TSDF itself host these capabilities? In instances where the TSDF hardware has the capabilities of general purpose machines, the TSDF itself could host these capabilities, otherwise, it could not. The extent to which hosting should be carried in the TSDF design procedure is unknown. One could propose a completely hosted TSDF, but hardware experience would not be gained.

Hosting is definitely an attained capability well within the current state of the art. Large scale computer systems come with with data busing, full operating systems, compilers for many languages, and many other capabilities. Transference of software from machine to machine can be accomplished readily with considerably less effort than in the past. Usage of a modern commercial data processing system in TSDF would certainly provide hosting capabilities. (Alternatively, a large scale data processor capable of emulating a modern commercial machine could also provide this capability.) Joint use of the TSDF by hosted generation and target architecture emulation proposed to limit costs and seems reasonable.

2.6.4 Users - The user class is still undefined. In thinking about TSDF functional capabilities the question of who the TSDF user is continually arises. Current commercial data processing systems support many different users through the software and environment they provide. However, few, if any, of these users emulate a new system architecture and none measure the performance of the emulation (more often simulation). Handling of many different users in itself is within the bounds of current technology but not for system emulation: the current technology provides the same machine capabilities for all users. It is recommended, however, that the user class be made to evolve as the capabilities of the TSDF evolve, that the initial user class be defined as system researchers who are interested in studying subsets of single machine technique. Evolution in more complete single machine operation and in different users should follow as capabilities are built and formed into a data base.

2.6.5 System vs. Machine Architecture - It is unclear whether the TSDF should be projected for developing a system of computers, a system comprised of any single computer, or a new system architecture for a new computer. It is also unclear whether any TSDF functional capability can uniformly apply to all three instances.

The current state-of-the-art supports the development of both unit machines and system architecture through emulation and simulation. Digital Avionics Information System (DAIS), for example, emulates a class of aircraft systems with one architectural concept. (Any specific aircraft system would probably be a subset of the entire DAIS.) It is recommended that the parent TSDF embrace the wide scope of development and the sibling TSDF embrace the more narrow developments. It is felt that the technical capability to support the developments mentioned earlier to a deep level of detail is definitely far away and may never exist at all. However, it is felt that a development decomposition can be adequately supported in the 1980 time frame.

2.6.6 Performance - The objective of the TSDC is to develop performance-oriented systems. It is unclear whether the objective means the best possible, the best for a specified cost, or no more than what is required for a specified application. Cost, for example, has never been discussed, nor has technology development in the sense of high performance circuit designs.

Minimization of hardware to achieve a specified level of performance is different than the open-ended approach of obtaining the best possible performance without hardware constraint.

The crux of the performance issue seems as centered on logical performance as it is on execution speed, i.e., fielded systems do not perform well enough because they are fielded with faults that must be corrected. This, in turn creates a more costly development through additional software, which lowers performance through additional use of processing facilities. It is recommended that in the parent TSDF logical performance to be emphasized and in the sibling, execution performance be emphasized. By stressing logical performance the goal will be to develop a logically faultless system thus eliminating the costly redesign phase. (Note the word "goal.") Execution performance in the sibling provides "proof" that the system thus designed will meet projected performance goals.

2.6.7 <u>User Insight into the TSDF</u> - User insight depends on designing the target architecture with HLHDL and HOL concurrently with specifying detailed performance measurements, for example, which require intimate knowledge of internal TSDF details. It appears desirable that the user be given the capability of specifying at a uniform level all aspects of his TSDF scenario. This implies a High Level Performance Measurement Language (HLPML). The current technology in HLHDL is primitive; in HLPML it is nonexistent. A definite desire for an HLHDL capability has been present for some time, but little has emerged that can be labeled successful. It is recommended that an HLHPL and HLPML be defined at a level comparable to the HOLs. It is recognized that this is not yet supported by the current state of the art.

3. Reconfigurable Computer System Design Facility (RCSDF)

In this section, an evaluation of the RCSDF and attendant technical issues is presented, along with a discussion of operating philosophy, capabilities, and procedures. The RCSDF mirrors the TSDF for researching the technical feasibility of the TSDF concept.

3.1 Fundamental RCSDF Doctrines

Premises for the RCSDF which are felt relevant to scoping RCSDF activities are discussed in this section and represent a preliminary outline of the activities for the RCSDF evolving at RADC. These premises are stated as a framework for orderly growth in the development of the capability for the design facility and to provide insight into the direction of RCSDF development. A distinction is made between the user of the provided capabilities, the host that supports the user, and the RCSDF itself.

3.1.1 <u>Emulation vs. Architecture Development</u> - The independent development of a new computer architecture shall be subservient to the objective of emulating/simulating/testing a user-postulated new architecture. The formulation of a computer (or system) architecture is regarded as the responsibility of the user who describes his system to the RCSDF through high level languages. If the RCSDF developer is its own user (meaning that the personnel who staff RCSDF as developers act both in user and developer roles) and the distinction between user and RCSDF is not made, then the separation of user and developer becomes unclear, confusing user responsibilities with RCSDF responsibilities. Architecture in the TSDF concept through RCSDF research requires that a clear delineation of user responsibilities and developer responsibilities be maintained.

3.1.2 Research Vehicle - The intent of the RCSDF is to perform research in support of TSDC. The attainment of research objectives will not be limited by the practicalities of militarized equipment or its software equivalent, mil-spec software. Sensible requirements that reflect the research nature of the facility should be imposed for reliability, maintenance, repair, and documentation. Attention to support maintenance should enable research to be effectively performed in a timely manner. Existing equipments and software will be used whenever they are adequate.

3.1.3 User Accessibility to Equipment - Users shall have total access to RCSDF capabilities. This access shall allow modification of any item such as software, microcode, nanocode, configuration options, and performance measurements. Total access is needed in order to thoroughly understand cause-and-effects associated with the research performed.

3.1.4 System vs. Data Processing - In the near future, RCSDF will support data processor emulation/simulation research before doing either application systems research or general data processing systems research. Longer term support of the latter two items can be considered on a case-by-case basis as needs warrant it. Initially, it is important to limit the directions of development, since too much diversity would tend to dilute the effectiveness of the research. Allowing the scope of activities to broaden as comprehension develops permits orderly development.

3.1.5 Hosting - Development of software to operate on the RCSDF shall employ a host computer in preference to developing support software on the RCSDF emulation/simulation hardware itself. Support software functions are generally available as commercial software elements, eliminating the need for developing RCSDF-specific software.

3.1.6 RCSDF Operating Environment - Initially, users of the RCSDF shall operate under the control structure defined for the RCSDF. Operating systems which are intrinsic to specific processing elements of the RCSDF shall be made available to users under the defined control structure. Since it is the control structure which provides the reconfigurability, the efficiency of existing operating systems will be affected. However, users are free to create their own operating system(s) and architecture, which can be hosted on the RCSDF.

3.2 Emulation Operating Philosophy

The key to describing an emulation operating philosophy is to maintain a distinction between the facility (i.e., the RCSDF), the user, and his generation tools (host). With this distinction emulation development responsibilities can be assigned to either of these three entities in terms of the architecture evaluation process. This distinction applies to both the TSDF and the RCSDF, and therefore emulation studied in RCSDF would be extendable to the TSDF. This distinction is used to homographically map the RCSDF to the TSDF and does not preclude the RCSDF (and its developers) from being users.

The user (human subsystem), host (hosting subsystem), and the facility (reconfigurable subsystem) have been generally described previously in paragraph 2.4, and those descriptions and ideas are pertinent to the discussions presented in this section.

The emulation operating philosophy is based on the fact that the facility provides services analogous to those provided by a computer center. However, RCSDF service is different in that it concentrates on architectural evaluation and development rather than on the normal program production. Programming and software are assuredly involved, but the end objective appears not to emphasize the program execution services but the evaluation of the system described and operation as specified by the user. If the RCSDF is thought of as a service facility, the services it provides can be described and its responsibilities delineated. Likewise, if a user wishes to solve his system design problem and wishes to use the RCSDF, his responsibilities can also be described and delineated. It is understood that user responsibilities cannot always be fully described for each and every user in the user class because that class is undefined and users can be expected to vary. However, some responsibilities can be described with sufficient detail to indicate generally what users will need to do in order to perform their architectural evaluations on the RCSDF.

First we will outline a generalized emulation procedure. An RCSDF user who wants to solve a class of problems must first specify his problems in very precise forms using some sort of requirements specification languages. He then has to translate his software requirements into HOL description, and his hardware requirements into EDL. He also needs to specifically define the user environment that his system will be encountering, for example, the number of users that will be on his system, the maximum (and average) data rate for the I/O channels, and the minimum response time.

Using this information, the hosting system will generate an emulated version of the hardware architecture defined by the user and the software package of his application for the computer system he defined. Moreover, the scenario and test environment specified by the user will be produced.

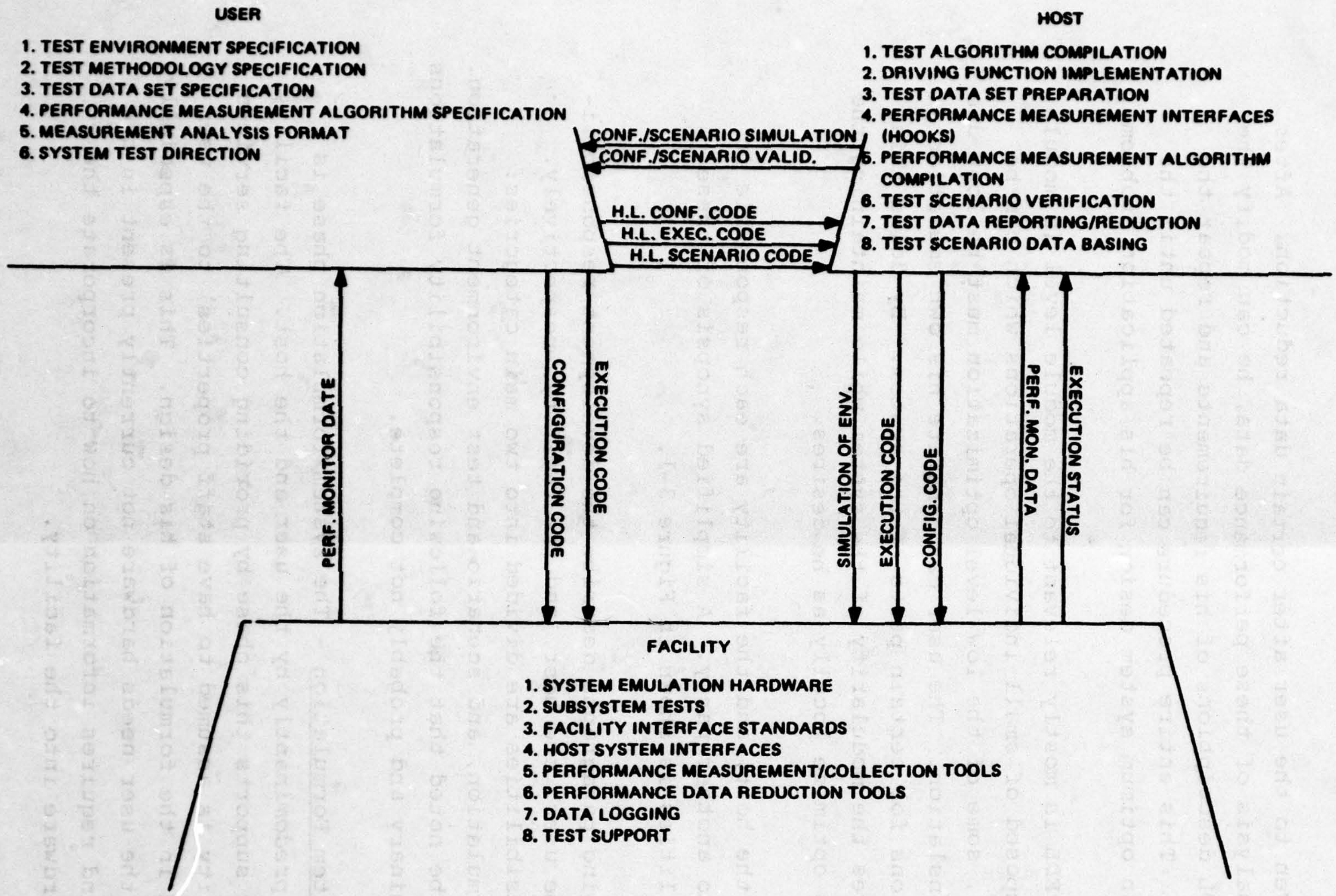
All of this information will be passed on to the RCSDF system. The execution will be emulated, and various performance measures be monitored. The performance measurement results will be the output given to the user after certain data reduction. After careful analysis of these performance data, he can modify the HOL and EDL descriptions of his requirements and repeat the procedure. This entire procedure can be repeated until the user has an optimum system design for his application problems.

Since the EDL is mostly relevant to the module level (a module may be composed of small individual operations which can be optimized), some of the low level optimization must be lost due to the translation. The user can generate his own emulation code sections for certain parts of the system. By doing this, he preserves the modularity of the system while maintaining the freedom to optimize locally as he desires.

The user, the host, and the facility are each responsible in some way to another party. A simplified synopsis of these responsibilities is shown in Figure 3-1.

The following paragraphs describe the development responsibilities of the user, the host, and the facility, respectively. The responsibilities are divided into two main categories: system formulation, and scenario and test environment generation. It should be noted that the following responsibility formulations are preliminary and probably not complete.

3.2.1 System Formulation - The system formulation phase is performed predominantly by the user and the host. The facility indirectly supports this phase by providing consulting services (the facility is assumed to have staff properties) to the user to aid him in the formulation of his design. This is especially true when the user needs hardware not currently present in the facility and requires information on how to incorporate the desired hardware into the facility.


USER
1. TEST ENVIRONMENT SPECIFICATION
2. TEST METHODOLOGY SPECIFICATION
3. TEST DATA SET SPECIFICATION
4. PERFORMANCE MEASUREMENT ALGORITHM SPECIFICATION
5. MEASUREMENT ANALYSIS FORMAT
6. SYSTEM TEST DIRECTION
HOST
1. TEST ALGORITHM COMPILATION
2. DRIVING FUNCTION IMPLEMENTATION
3. TEST DATA SET PREPARATION
4. PERFORMANCE MEASUREMENT INTERFACES (HOOKS)
5. PERFORMANCE MEASUREMENT ALGORITHM COMPILATION
6. TEST SCENARIO VERIFICATION
7. TEST DATA REPORTING/REDUCTION
8. TEST SCENARIO DATA BASING
CONF./SCENARIO SIMULATION
CONF./SCENARIO VALID.
H.L. CONF. CODE
H.L. EXEC. CODE
H.L. SCENARIO CODE
PERF. MONITOR DATE
CONFIGURATION CODE
EXECUTION CODE
SIMULATION OF ENV.
EXECUTION CODE
CONFIG. CODE
PERF. MON. DATA
EXECUTION STATUS
FACILITY
1. SYSTEM EMULATION HARDWARE
2. SUBSYSTEM TESTS
3. FACILITY INTERFACE STANDARDS
4. HOST SYSTEM INTERFACES
5. PERFORMANCE MEASUREMENT/COLLECTION TOOLS
6. PERFORMANCE DATA REDUCTION TOOLS
7. DATA LOGGING
8. TEST SUPPORT

Emulation System Evaluation

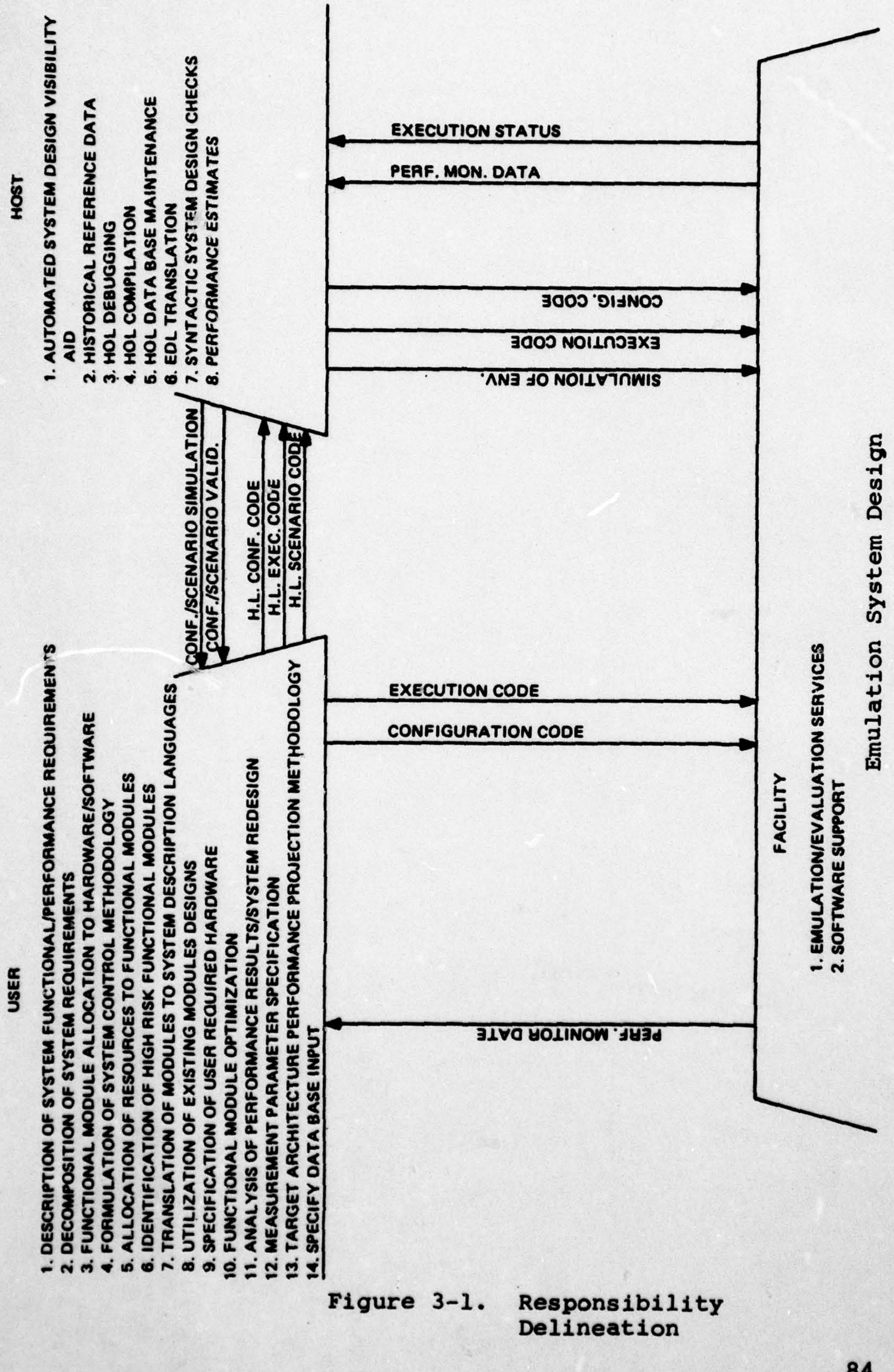

Emulation System Design
USER
1. DESCRIPTION OF SYSTEM FUNCTIONAL/PERFORMANCE REQUIREMENTS
2. DECOMPOSITION OF SYSTEM REQUIREMENTS
3. FUNCTIONAL MODULE ALLOCATION TO HARDWARE/SOFTWARE
4. FORMULATION OF SYSTEM CONTROL METHODOLOGY
5. ALLOCATION OF RESOURCES TO FUNCTIONAL MODULES
6. IDENTIFICATION OF HIGH RISK FUNCTIONAL MODULES
7. TRANSLATION OF MODULES TO SYSTEM DESCRIPTION LANGUAGES
8. UTILIZATION OF EXISTING MODULES DESIGNS
9. SPECIFICATION OF USER REQUIRED HARDWARE
10. FUNCTIONAL MODULE OPTIMIZATION
11. ANALYSIS OF PERFORMANCE RESULTS/SYSTEM REDESIGN
12. MEASUREMENT PARAMETER SPECIFICATION
13. TARGET ARCHITECTURE PERFORMANCE PROJECTION METHODOLOGY
14. SPECIFY DATA BASE INPUT
HOST
1. AUTOMATED SYSTEM DESIGN VISIBILITY AID
2. HISTORICAL REFERENCE DATA
3. HOL DEBUGGING
4. HOL COMPILATION
5. HOL DATA BASE MAINTENANCE
6. EDL TRANSLATION
7. SYNTACTIC SYSTEM DESIGN CHECKS
8. PERFORMANCE ESTIMATES
CONF./SCENARIO SIMULATION
CONF./SCENARIO VALID.
H.L. CONF. CODE
H.L. EXEC. CODE
H.L. SCENARIO CODE
EXECUTION STATUS
PERF. MON. DATA
CONFIG. CODE
EXECUTION CODE
SIMULATION OF ENV.
EXECUTION CODE
CONFIGURATION CODE
PERF. MONITOR DATE
FACILITY
1. EMULATION/EVALUATION SERVICES
2. SOFTWARE SUPPORT

Figure 3-1. Responsibility Delineation

2

3.2.1.1 User Responsibilities

3.2.1.1.1 Description of System Functional and Performance Requirements - The user is responsible for generating a literal description of the initial concepts of what the system must do. A requirement specification language is a good example of a formulation used to describe the system functional and performance requirements. This description will provide a structural definition of the system which is decomposable into functional modules. The level of description is equivalent to that normally provided in a system performance specification.

3.2.1.1.2 Decomposition of System Requirements into Functional Modules - The user should identify, isolate, and describe his system functional modules. He should consider such interrelationships of co-existing modules as:

potential of parallel execution
shared memory
data dependency
mutual resource demands
intermodule interface standards

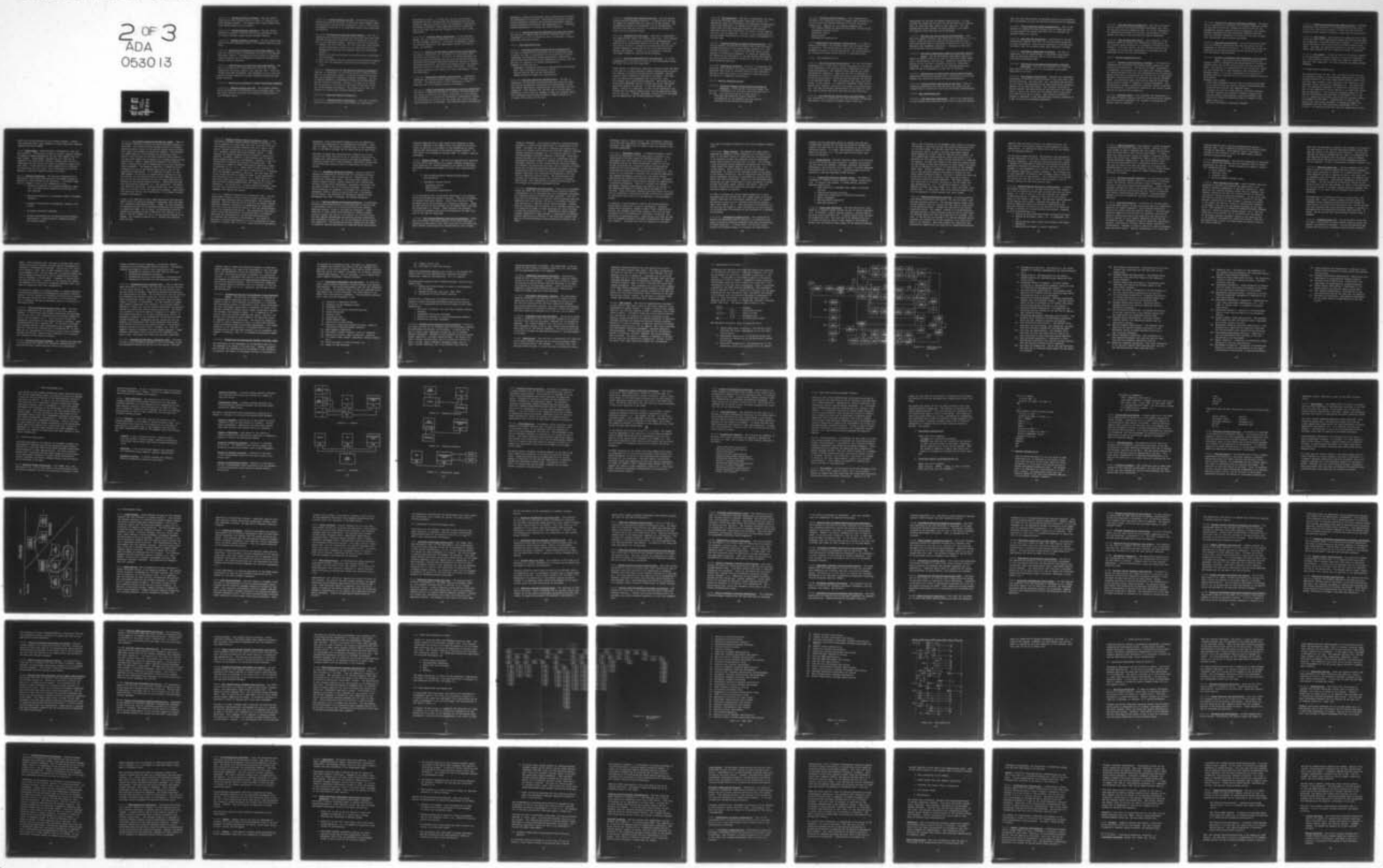
3.2.1.1.3 Allocation of Functional Modules to Hardware or Software Implementation - For each functional module, the user should tentatively decide whether to implement it in hardware or software.

3.2.1.1.4 Formulation of System Control Methodology - The user must determine the system structure and establish the control hierarchy by describing the interconnectivity of the defined functional modules. He must also define the rules for scheduling, sequencing, and control synchronization of the interconnected functional modules.

AD-A053 013 SPERRY UNIVAC ST PAUL MINN DEFENSE SYSTEMS DIV F/G 9/2
RECONFIGURABLE COMPUTER SYSTEM DESIGN FACILITY INITIAL DESIGN S--ETC(U)
JAN 78 D R ANDERSON, L D ANDERSON, K Y WEN F30602-76-C-0355

UNCLASSIFIED RADC-TR-78-6-VOL-2 NL

2 OF 3
ADA
053013



3.2.1.1.4.1 Hardware-Hardware Interface - The user should describe how each hardware module is going to interface with other hardware modules. The communication links between hardware modules should be well defined.

3.2.1.1.4.2 Software-Software Interface - The user should describe the control hierarchy and the data parameters used in control transfers among various software modules.

3.2.1.1.4.3 Hardware-Software Interface - The user should also specify the data parameters used in all hardware-software interfaces.

3.2.1.1.5 Allocation of Resources to Functional Modules - The user must define resource requirements for each of the defined functional modules. Consideration must be given to common or shared resources and the frequency of potential conflict with mutually defined resources.

3.2.1.1.6 Identification of High Risk Functional Modules - The user is responsible for identifying all high risk (or least understood) functional modules, both hard and soft. This will include prototype development (see paragraph 3.2.1.1.7 and verification through repetitive isolated testing utilizing the emulation facility.

3.2.1.1.7 Translation of Modules to System Description Languages

3.2.1.1.7.1 Hardware Modules into EDL - The hardware modules should be described in an Emulation Design Language (EDL). The EDL should contain structural and functional descriptions of each hardware module.

3.2.1.1.7.2 Software Modules into HOL - The software modules should similarly be described in some High Order Language (HOL). The description may be given in parametric form to test out the intermodular structure or in algorithmic details for algorithmic verification.

3.2.1.1.8 Utilization of Existing Module Designs - The user is responsible for utilizing previously constructed functional modules and creating the environment that is equivalent to or similar to the deployable system. The units that can be used are:

- o Modules that are compatible with the functional requirements, provide performance capabilities as identified during the decomposition process, and currently exist in a maintained library.
- o Modules that were defined and constructed during the decomposition process.
- o High risk modules that were previously constructed and tested.
- o Units that are designed to provide system simulation functions.

3.2.1.1.9 Specification of User-Required Hardware Attachments - The user is responsible for specifying special hardware attachments to the deployable system, including necessary tailoring to meet the standardized hardware attachment interfaces. The use of special hardware is implemented through an EDL description and HOL function references. Finally, the user has to verify the correctness of the data that are being transferred in and out of the hardware attachment. This allows him to test the validity of a previously untested hardware module.

3.2.1.1.10 Functional Module Optimization

3.2.1.1.10.1 Hardware Module Optimization - The user is responsible for optimizing emulation code beyond the optimization provided by the host. In doing this the user also assumes responsibility for verifying that the optimization he has performed is logically correct within the framework of operations he intends to perform. These optimized emulation codes will have to be input into the host together with the rest of EDL description.

3.2.1.1.10.2 Software Module Optimization - In an analogous manner, the user is also responsible for the optimization of his HOL language compilation. He should be prepared to modify his software modules at the (deployable) machine instruction level, if necessary.

3.2.1.1.11 Analysis of Performance Results and System Redesign - The user is responsible for the analysis of performance data collected during emulation. From this, he should determine the potential system bottlenecks and the major design deficiencies. He should then redesign identified functional modules to improve individual efficiency, and modify the system control and resource sharing structure to improve his system performance. Finally, he should reconstruct the system with these improvements, retest, and reevaluate.

3.2.1.1.12 Measurement Parameter Specification - Measurement parameters are the performance measurements to be made during the test. The specification of these parameters is used to guide the use of performance monitoring equipment.

3.2.1.1.13 Target Architecture Performance Projection Methodology - Projection of measured performance to predict the performance of the target architecture may require modification of the measurement parameters to reflect the differences in performance between the postulated architecture and the system upon which the postulated architecture is emulated. This requires target architecture parameters, predictive algorithms, and a method by which projected performance can be derived. Performance projection includes both factorization (deleting from measured performance general purpose emulation efficiencies) and enhancement (i.e., modifying measured parameters to account for faster hardware).

3.2.1.1.14 Specifying Required Reference Data Base Input Items - The user has to specify what historical reference data items (paragraph 3.2.1.2.2) he intends to store, conforming to certain guidelines given by the host.

3.2.1.2 Host Responsibilities

3.2.1.2.1 Providing Automated System Design Visibility Aid - The host is responsible for providing tocls and automated aids to assist the user in the production and maintenance of system (subsystem) and functional module requirements, performance descriptions generated during the decomposition process, and the decision for hardware or software implementation.

Examples of these aids are:

Management information relating to system design and development (such as progress and current status).

Editing help for HOL and EDL descriptions.

Simple simulation support for verification.

3.2.1.2.2 Providing Historical Reference Data - The host is responsible for providing and maintaining the historical reference data on performance requirements and measurement results, user-design (deployable) configurations, and available facility components. This implies that the host has to provide basic guidelines for the user who intends to store his reference data. This information will either be used to support emulation or to report test results to the user.

3.2.1.2.3 Providing HOL Debugging Facility - The host should provide some HOL debugging facility. It should allow the user to compile and execute some test data on his HOL program. The debugged HOL program will eliminate a lot of confusion and facility time that will arise if an untested HOL program is being run on an emulated system.

3.2.1.2.4 Compilation of HOL Input - The host is responsible for compiling HOL inputs into machine level code. This requires a description of the machine configuration and repertoire, possibly as an intermediate product of the EDL translator. The host should be able to integrate the HOL source with all the library components referenced and also with user-optimized machine code modules. Finally, it is necessary to provide the proper linkage for various software modules.

3.2.1.2.5 Data Base Maintenance for HOL Compiler - It is the responsibility of the host to store all the intermediate tables and codes during the HOL compilation.

Since one set of application algorithms (written in HOL) may have to be test-run in many different machine architectures and since the compiler is generally architecturally dependent, the HOL compiler should probably be designed in two parts. The first part is architecture-independent and include such items as a scanner and a parallelism recognizer. The second part is architecture-dependent and will involve the actual code generation and resource allocation. If the host stores all the phase 1 results in a data base, it will not be necessary to go through the two compiler phases every time the application algorithm is test-run. Instead, the phase 1 results can be retrieved and the compilation can proceed directly to phase 2.

3.2.1.2.6 EDL Translation - The host is responsible for translating user hardware requirements (described in EDL) into appropriate codes for establishing the deployable system configuration and repertoire. It should also be able to integrate these codes with user-optimized emulation code sections. Necessary architecture information for phase 2 of HOL compilation will also be generated. The host is also responsible for storing system descriptions at various steps of development in its data base.

3.2.1.2.7 Syntactic Checks for System Design Functions - Both the HOL compiler and the EDL translator should provide syntax correctness checks for the system design language (i.e., HOL and EDL) constructs. It should be noted that since the HOL compiler is architecture-dependent, it should be responsible for detecting any discrepancy in the EDL and HOL descriptions. Any errors detected should be reported to the user in order to avoid erroneous emulation.

3.2.1.2.8 Performance Estimates - The host is responsible for providing rough estimates for target architecture, as required. These may come either as estimates based on past experience (stored in host data base) or as analytical extrapolation of test data of a comparable architecture.

3.2.1.3 Facility Responsibilities

3.2.1.3.1 Providing Hardware System Emulation/Evaluation Expertise - During system formulation phase, the facility is responsible for providing consulting services to the user. The consulting may be in areas such as:

Efficient EDL coding for faster emulation
Incorporation of new hardware into the facility
Available system evaluation techniques

3.2.1.3.2 Providing Software Support - This responsibility includes a full operating system to support user evaluations of a minor nature plus special software to provide non-normal capabilities. Examples of such software supports are:

Test system loading and emulated environment initialization
Transparent process control
Peripheral control
Memory control
Intermachine communication

3.2.2 Scenario/Test Environment; System Testing - In order to observe performance, the emulated system must be tested while in operation. The construction of the test environment and subsequent testing of the emulated system, the second phase of the interactive system design process, is described below.

3.2.2.1 User Responsibilities

3.2.2.1.1 Test Environment Specification - The test environment specification is a high level, overall description of the tests to be performed on the emulated system, requirements for the system configuration, and subsystem tests required for system configuration components. Requiring the user to generate a specification (actually nothing more than a concise description of testing requirements) formalizes and disciplines the testing of the emulated system. Test environments may range from very simple to very complex. As complexity increases, formalization becomes more important because of the logical intricacies involved. The test environment specification is the governing document used to define testing to host and facility functions.

3.2.2.1.2 Test Methodology Specification and Development - The test methodology is essentially the test algorithms, procedures, measurements, and possibly validation used to meet the requirements of the test environment specification. It also includes the sequencing of tests and the delineation of requirements for post-test data reduction and reporting. The test methodology further refines the testing phase to the extent that specific test functions can be developed.

3.2.2.1.3 Test Data Set Specification and Development - Test data sets support the basic test algorithms of paragraph 3.2.2.1.2 above. These data sets are used by the algorithms to cause specified system behavior expressly structured for test and to determine data dependent performance. Sequences of data sets may be used for time-flow scenario.

3.2.2.1.4 Performance Measurement Reduction Algorithms Specification - The raw performance measurement data will generally require reduction in order to facilitate report generation, understanding, and analysis. Reduction algorithms are applied to the raw data to produce statistical, summarized, tabulated, or graphical reports of the observed performance. Reduction may be performed in either host or facility.

3.2.2.1.5 Specification of Measurement Analysis Report Format - Reduced and analyzed performance data is presented in prescribed formats that succinctly portray the desired parameters.

3.2.2.1.6 Directing and/or Performing System Tests - This is essentially performing the required tests and collecting performance data as specified in the test environment specification.

3.2.2.2 Host Responsibilities

3.2.2.2.1 Test Algorithm Compilation - This is the translation of user-defined test algorithms into a facility driving function code (the code which causes the emulated system to be stimulated for the purposes of test) and facility driving function sequences.

3.2.2.2.2 Driving Function System Implementation - This is the mergence of compiled driving functions into an operating stimulus environment that causes test data sets to stimulate the emulated system for testing.

3.2.2.2.3 Test Data Set Preparation - Stimulation of the emulated system requires data that the emulated system processes as input information. Data sets are generated to simulate system inputs that the emulated system would encounter in usage.

3.2.2.2.4 Performance Measurement Interfaces - The host is responsible for the establishment of performance measurement traps in generated emulation code to allow the capture of performance measurement data.

3.2.2.2.5 Compilation of Performance Measurement Reduction Algorithms - The host is responsible for compiling data reduction algorithms for the performance measurements specified by the user.

3.2.2.2.6 Test Scenario Verification - Separate and independent development of emulation system structure, test algorithms, test driving functions, test data sets, etc. introduces the possibility of logical and procedural errors that would create erroneous behavior in the emulated system. Test scenario verification brings together the system emulation and testing at the logical level to verify that errors are not present and to flag potential difficulties due to timing. The scenario verification process is analogous to the verifications currently performed as a part of high level language compilation.

3.2.2.2.7 Test Data Reduction/Reporting - The host is responsible for receiving test and performance measurement data from the facility and then compacting them. It is also responsible for generating the report to the user, and storing all intermediate test data in the data base for future references.

3.2.2.2.8 Test Scenario Data Basing - Once generated, a test scenario should be reusable for evaluating a modification of the original system emulation or a completely alternative architecture. To do this easily, the test scenario should be recreatable from a data base (preferably an automated one) and modifiable to suit the new architecture.

3.2.2.3 Facility Responsibilities

3.2.2.3.1 Providing System Emulation Hardware - The facility is responsible for providing the hardware system required for generalized system emulation. The design as needed, procurement if necessary, and integration of any additional hardware needed for system evaluation are also the responsibility of the facility. System architectural alternatives for testing will include single machines, multiple machines, and machines with special hardware added. Any particular system architecture desired by the user may be either a subset or superset of existing equipment. The configuring of hardware includes any necessary design to develop the configuration, interfacing of components, and actually implementing the desired system emulation.

3.2.2.3.2 Subsystem Tests - This validates the operability of the separate system components and the intercommunication between the interconnected components and includes the generation of special tests as required.

3.2.2.3.3 Provision of Facility Interface Standards - The facility is responsible for providing the interface standards between the host and the facility, between the facility and all its peripherals, between standard facility components, and between standard components and user-designated hardware additions. Facility interface standards do not restrict the user to specified interprocess communications techniques.

3.2.2.3.4 Host System Interfacing - The facility should establish the necessary connections with the host system for the acceptance of stimulation functions. It should also verify the compliance of the host-generated driving functions with the test environment specifications.

3.2.2.3.5 Providing Performance Data Measurement and Collection Tools - The facility should provide hardware and software tools for performance data measurements and collection. Hardware tools may include sensing probes and data recording media. The facility is responsible for the installation, checkout, and validation of these equipments when connected to the emulated system. It is also responsible for verifying the recording facility compatibility with the data reduction facility verification.

Software tools include:

A data base management function that will enforce efficient use of mass storage for interim storage of performance data collected during system and subsystem tests.

Processes that provide RCSDF operator interface functions and that interpret and respond to command language which provides control and monitoring functions entered at the start of or during the execution time of a system or subsystem under test.

Efficient performance measurement packages.

3.2.2.3.6 Supplying Performance Data Reduction Tools - Although performance data reduction is the responsibility of the host, the facility should also attempt to reduce the high volume of performance data wherever possible to help simplify the process.

3.2.2.3.7 Data Logging - The facility is responsible for a data log on all data items transferring between various facility hardware components, including special user hardware. It also has to record the system clock time at which the data transfer takes place. This data log should be available to the user as an emulation trace whenever the user suspects an error in emulation.

3.2.2.3.8 Test Support - Assisting users to the extent required in the performance of system testing is a facility responsibility. The assistance should include such items as consulting analyses, test conduction, and other coordination aspects deemed necessary.

3.3 RCSDF Facility Capabilities

The responsibilities of user, host, and facility were outlined in preliminary fashion in the preceding paragraphs to illustrate the partitioning of the Total System Design Process (TSDP) over the three major elements (user subsystem, host subsystem, reconfigurable system) identified in paragraph 2.4. The fourth element, the deployable subsystem, is essentially the product and has not been addressed in detail because it is essentially dependent on application. This following paragraphs expand on the facility responsibilities described in paragraphs 3.2.1.3 and 3.2.2.3, specifying and further describing the necessary technical elements of the RCSDF. "Capabilities" as used in this section are those things necessary to adequately support the responsibilities set forth previously (paragraphs 3.2.1.2, 3.2.2.3). It should be noted here that the capabilities listed below are only preliminary and by no means complete. However, they should provide a good framework to which additional required capabilities may be added.

3.3.1 System Design - System design is prominently a user and host activity. The facility is used to confirm, analyze, develop, or improve a system designed by the user with support from the host element. Nevertheless, because the user class can be expected to be quite broad, extensive capabilities must be incorporated into the facility in order to adequately support the user class. Simplification of these capabilities is possible by restricting the user class, but not advisable at this time because an important capability might be omitted.

3.3.1.1 Emulation Expertise - The facility responsibilities described in this section must provide the hardware system expertise for system emulation and evaluation. The general capabilities needed to fulfill this responsibility are:

- o Standard methodology for emulating sequential, parallel, and associative processing architecture for unit machines
- o Emulation packages for a reasonable number of standard computers
- o Standard interconnection methodology, hardware, and control
- o Peripheral emulation technique
- o Standard methodology for emulating multiple machine architecture - multiprocessors, federated systems, and network systems

3.3.1.1.1 Unit Machine Emulation Methodology (UMEM) - UMEM is the general technique by which the system emulation package is developed. The system emulation package converts the general purpose emulation system to the desired target system architecture, which then accepts the execution software. The methodology (UMEM) does not itself develop the architecture that a specific user may want but provides guidelines and methods to the user which enable him to produce his intended emulation with greater ease. UMEM is supported by examples of how specific kinds of architecture may be implemented using an EDL (Emulation Design Language). The examples supportive of UMEM would also illustrate the use of performance monitoring as part of the architecture formulation. It is important to point out that UMEM supports the division of responsibility between the user and facility in the design process, in that UMEM does not portray to the user the specifics of how the user architecture is emulated on the general purpose emulator. UMEM gives the user only the emulation constructs i.e., EDL constructs. The emulation specifics (microcode, interconnect patterns, etc.) are produced as a function of the EDL compilation process. These emulation specifics map the user architecture to the general purpose emulator architecture.

A user is not expected to have a deep familiarity with the process by which an emulation package is developed, hence the need for a emulation development pattern, a UMEM. As user expertise is developed variants of the standard methodology can be pursued to expand upon UMEM. As a minimum, the methodology should be supported by examples of how a sequential unit machine, (e.g., PDP-11, IBM 360, SPECTRA 70, etc.), an associative machine (e.g., STARAN), and a parallel machine (e.g., ILLIAC) were emulated using the EDL.

3.3.1.1.2 Standard Computer Emulation Packages (SCEP) - Standardization pervades systems designs because of economics. It can be expected that some users will want to pursue their systems designs using standard computers and will require (and expect) standard emulations to be available for their usage, e.g., Standard Computer Emulation Packages (SCEPs). A SCEP converts the general purpose emulation system to operate as a specific "standard" computer to some defined level of detail. (Many standard machines are configurable, therefore it may not be practical to emulate all possible configurations.) The building of a number of SCEPs contributes to defining of UMEM as described in 3.3.1.1.1. SCEPs should be made available to users in two forms: 1) a high level construct and 2) a compiled emulation package cataloged in the host. The high level construct allows the user to modify the SCEP to the configuration desired (memory size, I/O channels, etc). Modifications at this level would be accomplished by changing the EDL (this mandates an easily understandable EDL) and recompiling the EDL constructs. The second form, a library emulation package, frees the user from constructing an EDL description of the standard component and compilation thereof.

3.3.1.1.3 Standard Interconnection Methodology (SIM) - The general purpose emulation system should be highly alterable for emulating many different system architectures, both existing and proposed. Multiple machines may be used for the general purpose emulation hardware necessitating a methodology for interconnection that allows the emulation system to have the desired properties of the emulated architecture. This methodology (SIM) would govern how an EDL description is developed that 1) accurately describes the interconnect of the desired architecture and 2) minimizes the interconnect relationships. The latter is desirable if observed performance is to be extrapolated or interpolated or factored (see 3.2.1.1.13) using estimated architecture parameters, since interconnect simplicity will be required to minimize the complexity of the extrapolation procedure. Examples to support the SIM are a necessary part of this capability and, to the extent pursued, may be usable per se by some users.

SIM would be based upon interconnection capabilities in the facility and would include the use of available interconnection hardware and interconnection control and handling packages. EDL constructs would link to these standard interconnect capabilities and the SIM would guide the user in formulating the EDL constructs.

3.3.1.1.4 Peripheral Emulation Technique - Certain classes of systems are dominated by interactions with peripheral devices, typically displays, mass storage devices, communications, and card/printer input/ouput media. Emulating these systems on a general purpose emulator requires expertise and a methodology by which the particular peripherals, peripheral interfaces, and peripheral handlers can be accurately portrayed. Included in this methodology are examples of techniques for simulating peripheral loading, making one particular kind of peripheral mirror another, and modifying the information passed to/from peripherals such that it emulates the desired peripheral.

3.3.1.1.5 Multiple Machine Architecture Emulation - Increasingly, multiple machine architecture is being employed to achieve higher performance levels, as in distributed processing systems, federated systems, networks, and parallel processors. Representation of these system architectures on a general purpose emulation system that may be a unit machine is a distinct problem that requires methodology for handling inherent parallelism and intermachine interaction, as well as emulation of the instruction repertoire. The parallelism problem coupled with the need to translate observed performance to expected target architecture requires expertise that a user could not be expected to have. A user is expected to be expert within his system architecture, while the facility is the expert on the methodology of representing that architecture on the general purpose emulator. To some extent this methodology overlaps UMEM, for example, parallel processors.

3.3.1.2 Support Software - The facility responsibility described in this section must provide the support software necessary to allow execution of a system designed to be tested on the RCSDF. The general capabilities needed to fulfill this responsibility are:

- o Test system/subsystem loading and environment initialization
- o Transparent process control
  - Peripheral control
  - Memory control
  - Intermachine communication

It is anticipated that the processes described in this section would be developed and tested on the host. They can be hardware or software functional modules. Source data versions of the processes should be maintained in a library on the host. Object or code versions of the processes can be constructed on the host and maintained in a library. These versions are processor architecture dependent and could be developed at the same time each of the SCEP are developed.

3.3.1.2.1 Test System/Subsystem Load and Initialization - Hardware and software functional modules of the system that were designed and developed on host facilities must be collected with their defined interconnectivity characteristics into a single system or subsystem. This process produces a system/subsystem recorded on a mass storage medium that is transportable to the RCSDF. Processes are required to establish the system architecture by loading micro-instruction functional modules, standard computer emulation packages, etc. into control memory of the microprogrammable processor elements of the RCSDF. Functional modules recorded with the system will be transferred into the configured processor element or elements by the load process. Subsequent execution of some initialization functional modules will perform initialization functions, such as enabling I/O data paths with configured standard/nonstandard peripheral devices, establishing rules and limits for memory and mass storage management algorithms, and establishing communication with the facility operation control center. The load processes will transfer the functional modules that constitute the system being tested, thus establishing an execution environment for the testing of an emulated deployable system.

3.3.1.2.2 Transparent Process Control - There will be processes within the RCSDF that will provide facility resources control. A methodology will be developed that will standardize interfaces for components requiring RCSDF provided resources. This will allow for the development of resource control processes that are compatible with and optimize the use of the hardware architecture selected for a specific system being developed. Rigidly enforced interfaces and utilization of the standard resource control algorithm provided in these processes will allow the user to alter the configuration, i.e., add/subtract memory modules, increase speed/size of mass storage, or add/subtract processors without impacting the structure of the functional modules that make up the system/subsystem being emulated and tested on the RCSDF.

Peripheral control, memory control, and intermachine communications have been identified as candidates for facility-provided processes, and capabilities included in each are defined in the following paragraphs.

3.3.1.2.2.1 Peripheral Control - A standard peripheral interface will be developed to allow many types of systems to be constructed on the host using one interface protocol. It is anticipated that this capability will allow simulation of peripherals during the system development stages on the host, as well as provide for a consistent method of establishing guidelines for the development of emulated peripherals or interfacing directly with those provided at the facility. Processes will be required to translate the information handled by the standard peripheral interface into the actual formats required by the configured or emulated peripherals. Processes will also be required to functionally simulate the capabilities of emulated peripherals. They will interpret the standard peripheral interface requirements and simulate the effects of the expected peripheral on another device provided at the facility. Data volume, data transfer rates, and control functions will be simulated to effectively provide a functional simulation of the actual device that is to be employed in the deployable system.

In addition to the peripherals used by the system/subsystem being tested, there will be sets of peripherals - possibly consisting of processors, magnetic tapes, disks, communication devices, and standard input/output devices, that will be used principally for the presentation, extraction, collection, and production of performance data. These devices will also be used via the standard peripheral interface introduced by the system test scenario. Processes will also be required to translate this type of standard interface to the actual hardware requirements.

3.3.1.2.2.2 Memory Control - The demands for memory space expected for potential systems designed and configured to execute on the RCSDF cannot be established and correspondingly, it is therefore assumed that adequate directly addressable memory space cannot be provided for all cases. This creates a need to provide memory management processes that will effectively extend the memory space as seen by the user, i.e., virtual memory. When a standard interface methodology is implemented, the system can be developed without being aware of the memory limitations, and the performance data generated during subsequent executions on the RCSDF will provide information that can be evaluated. Results of the performance data evaluation can lead to decisions to alter the memory or backup storage size and/or speed, rebuild, and retest. Thus the testing, evaluating and redesigning provides a means of optimizing memory utilization.

In order to provide the virtual memory capability to users, processes must be provided that will manage memory utilization (paging techniques and relocatable segmentation, for example) and maintain control of the interface with backing storage (disk, bulk memory, etc.). These processes are transparent to the user but the impact of their intervention during execution on the RCSDF must be accounted for and provided for in performance analysis.

3.3.1.2.2.3 Intermachine Communication - The communication between two deployable modules should not be hampered by the types of emulation hardware components that they are executing on during emulation. In other words, the facility should provide the software process to transform the communication

message from one deployable module to another into the form transportable between the two physical emulation hardware components, and then back to the form receivable by the receiving deployable module. All these transformations should be transparent to the user, i.e., the user need not know about such activities going on.

3.3.2 <u>System Testing</u> - The most important phase of the facility activity is during the system testing. Providing and maintaining the evaluation hardware, interface standards, performance measurement, and collection tools are some of the more important responsibilities of the facility. Extensive capabilities are essential to provide meaningful RCSDF services.

3.3.2.1 <u>Generalized Emulation Hardware System</u> - The RCSDF is intended as a research vehicle capable of general system emulation. To achieve that purpose, it needs certain intrinsic hardware capabilities:

- **Easy operation on embedded state images of emulated machines**
- **Generalized decoding structure**
- **Simplified configuration of emulated environment**
- **Memory management**
- **Multiple-machine emulation**
- **Data path emulation**

3.3.2.1.1 <u>State Image Operations</u> - The data and control state images of a computer system include the set of working registers (e.g., accumulator, general purpose registers, program counter, and other status registers) and the memory system where the data and program are held. The first job of an emulation facility is to map the data and control state images of the emulated machine into the existing state images of the emulating machine.

Then it has to operate on the embedded state images of the emulated machine in the same way that the emulated instruction does on its state images. To perform these two jobs efficiently, the facility should be able to provide easy image mapping. Image mapping for a small class of machines may be simple, but for a generalized system emulation, where the systems to be emulated can vary greatly, great systems flexibility is required. For example, the data word length of the emulating system should be wide enough so that each reference to state image to be emulated can be emulated with one or two references to the embedded state image. However, embedding a short emulated machine word into a long emulating machine word would be inefficient and would require excessive memory. How best to handle the resulting situation is the kind of decision that continually faces a generalized emulation system developer. One useful feature that would help an efficient emulation is the multidata word size addressibility, i.e., the emulating machine is able to address different size data images (e.g., byte, half word, word, double word). In some cases, some shift and mask capabilities will be necessary to access unusual size state images.

3.3.2.1.2 Generalized Decoding Structure - To emulate generalized system architecture efficiently, the facility needs a generalized decoding structure. The emulated program output from the HOL compiler need not contain the same type of instructions as the emulated machine. Instead the instruction may contain fixed-size fields. For example, instead of allowing different emulated machines to have different sizes for the op code field, the facility should use only one fixed size for op codes. However, if the facility is allowed to run genuine emulated machine programs (not produced by the HOL compiler), this fixed field idea is not feasible. Then some flexible bit extraction and manipulation capabilities are required for generalized decoding.

Features like the barrel shifter, the masking function, and data insertion in arbitrary fields are good examples of such capabilities. An associative memory will also be very helpful for the decoding function.

For efficient system emulation, the emulation code produced by the EDL translator (in the host) should be in the form of parameterized templates. The facility should then provide a means of dynamically modifying the emulation code semantics based on parametric information extracted from the emulated instruction. Examples of such parameters are the shift count, ALU functions, and indirect addresses of general registers. For efficient use of such templates, providing a case statement type of emulation instruction will be very helpful. This instruction will provide the ability to test several conditions and branch to any of several sections of code which service them.

3.3.2.1.3 Emulated Machine Environment Configuration - A machine environment consists of (1) the data and control state images, (2) a set of primitive actions used to modify and test the state images, (3) a set of control rules which decides the sequence of primitive actions to execute on the basis of the current status of the control state image. The emulation system operates on two machine environments: emulated and executing. The facility should be capable of statically reconfiguring the executing machine environment for the duration of an emulation, so that it matches the emulated machine environment. Examples of such reconfigurations are:

- o Setting up gating patterns between registers and buses
- o Setting up arithmetic modes, i.e., ls complement, BCD, etc.
- o Setting up data word lengths for arithmetic and memory operations
- o Specifying the number of general registers.

3.3.2.1.4 Memory Management - The problems of memory management of generalized emulation system are twofold. Because of the large number of control programs that are needed, the emulation facility should have a large control store system. In cases where very large control stores are needed, a feasible solution may be the conventional hierarchical memory system coupled with virtual addressing capabilities. The second problem occurs when the emulated machine has a hierarchical memory system. The facility should then have the capability of mapping each emulated memory level into the emulation facility memory system. Hence, multilevel memory system and complex address mapping functions are needed.

3.3.2.1.5 Multiple-Processor Emulation - To efficiently emulate a multiple-processor system, such as a distributed or SIMD-type system, the facility should provide at least a subsystem consisting of some fully interconnected processors. Larger multiple-processor systems will have to be emulated by replication, and systems with limited interconnections will have to be emulated with certain interconnection path disabled. Multiple memory modules will also be needed for efficient multiple-processor emulation.

3.3.2.1.6 Data Path Emulation - To emulate all the data paths in the emulated system, the facility should have full connections among all its system components. Only selected connections would be activated during any specific emulation. The data path emulation capability should be able to represent explicit and implicit architectural paths both explicitly and implicitly in order to provide the full range of emulation levels that may be needed by users. Examples of explicit paths are input/output channels, communication networks, busing system, and matrix interconnects. Examples of implicit paths are internal processor buses and memory buses. Explicit representation of a data path requires physical path; implicit representation implies a physical path. Data traversing an explicit path actually moves through the data path. With implicit paths, time delay is ascribed to the traversal without the data actually moving through a physical data path.

3.3.2.2 Subsystem Testing - The facility responsibility described in this section must perform facility subsystem tests to assure operability for testing. General capabilities needed to fulfill this responsibility are

- o basic hardware tests
- o emulation tests
- o interface tests
- o facility-supplied software tests

3.3.2.2.1 Basic Hardware Tests (BHT) - Basic hardware tests are performed to validate and diagnose the operability of the emulation hardware independent of emulation constraints. BHTs apply to both standard hardware (normal facility hardware) and hardware created by the user for inclusion within the facility (exclusive of emulation described hardware). In the latter case BHTs may be created by the facility as a function of the user problem (created when the user attempts to emulate his system), in the former case BHTs are created or obtained when hardware is included in the facility and becomes universal to the user class. When hardware is a recognized commercial product (display, tape unit, commercial general purpose machine, etc.), BHTs are either part of the product and or obtainable with the product. For developed hardware (hardware created by the facility that is not generally obtainable as a commercial product), BHTs are developed during or after the design phase and become a part of the facility BHT set.

BHTs that are developed for special user hardware are created in the same manner as for facility developed hardware except that they are unique to the specific user emulation. The facility develops the special hardware BHT as a part of integrating the user specified hardware with the general facility hardware. This familiarizes the facility with the nonstandard hardware and its behavior, as well as developing a hardware checkout tool.

3.3.2.2.2 Emulation Tests (ET) - ETs are component-level tests used to check the general purpose emulation system components after being configured to perform the intended emulation. These tests emphasize emulation operability and as such apply only to components that are altered (by microcode, for example) from emulation to emulation. (Unalterable components can be tested with BHTs). ETs are created as an emulation function by the facility each time the user decides to create a new architecture. If the user selects a standard computer emulation package (see paragraph 3.3.1.2) then a ET will probably exist and not be generated.

The principle of an emulation test can be described in the following way: given a general purpose emulation that can implement various repertoires and architectures through the installation of instruction/configuration code, the ET verifies that the installed code causes the base hardware to operate in accordance with the emulation functional specifications. In effect, the ET verifies that the generated emulation code is operable.

3.3.2.2.3 Interface Tests (IT) - ITs are standard tests that are used to verify data transfer between emulation system components (interface-to-interface). ITs can and should be separated into unit-to-unit interface tests and path interface tests. Path interface tests are used to validate data transferability from one interface through several intermediate interfaces to a destination interface. Unit-to-unit ITs, which are a subset or lower level of path interface tests, validate a path segment only. ITs are used when the general purpose emulation system is brought up or when a suspected error occurs that is traceable to a data path. A separate IT should exist for every distinct path. However, a test program may combine the separate ITs to provide an Integrated Interface Test (IIT) which is unique for each user configuration.

ITs range from simple (I/O channel checks) to complex depending on the path architecture and embedded path processing function. However, ITs are limited to validating only correct bit transfer. This excludes validating the interfacing of the transferred bits to resident destination processes, which is a function of the system tests.

3.3.2.2.4 Facility-Supplied Software Tests (FST) - The user emulation experiments can be expected to vary from simple to complex. In those cases where simple experiments are performed the user may opt to use a standard facility software capability (an OS, an I/O handler, etc.). FSTs are used to verify the operational correctness of the standard facility software. Each standard software has a separate FST. In the case of configurable standard software, each configuration has a separate FST. Like ITs, FSTs can be combined into an Integrated FST, an IFST. FSTs are not responsible for verifying correct usage of standard software by the user; they are only responsible for validating the integrity of the standard software.

3.3.2.3 Facility Interface Standards - The capabilities described in this section must provide the interface standards between host and facility, between facility and all its peripherals, between standard facility components, and between standard components and user-designated hardware additions. The general capabilities needed to fulfill this responsibility are:

- o Standardized interface with host (driving function)
- o Standardized peripheral interfaces
- o Standardized facility intercomponent interconnections
- o Standardized interface for user-designated hardware

3.3.2.3.1 Standardized Host Interface (SHI) - Host as used here means essentially the element that provides the driving function for the system emulation. This element may be different than the host which, for example, compiles the HOL/EDL into execution and system definition codes. The interface with the host includes both hardware and software connections, as well as the fundamental methodology used to cause the emulation system and host system to interact in synchronism. The SHI consists of compatible channel connections (I/O channels), the software channel handlers at either end, and the linkage methodology for emulation software to emulate the channel desired connection. The emulation software to cause the channel to appear as it appears in the target architecture is the user responsibility, and it is generated to operate in the framework of the facility provided linkage methodology. Compatible channel connections are the hardware interconnections that allow unmodified move without error between the driving and emulation system. The software handlers provide the error control, linkages, etc. that are needed to transfer information on real digital channels. These handlers investigate status, perform actions dependent upon status codes, and handle both normal and abnormal conditions.

3.3.2.3.2 Standardized Peripheral Interfaces (SPI) - The facility will utilize peripheral equipments that are probably unlike the peripherals desired by the user for his system. The real channels used to connect the emulating hardware to these peripheral equipments will need to be transformed so that they operate as the peripherals of the user architecture. This is done with SPIs, which may be software, microcode, or a combination of both. It is improbable that SPIs will be hardware because of the expense and difficulty of modifying/adding additional interfaces to the emulating system. SPIs are emulation components for standard channel interfaces that are available in much the same way as SCEPs (3.3.1.1.2), typically, Mil-Std-188, RS-232, and Mil-Std-1397.

3.3.2.3.3 Standardized Facility Intercomponent Interconnections (SFII) - The facility is assumed to be composed of multiple computing machines that are connected in a specified number of ways to allow different architectures to be emulated. The interconnect pattern is defined by a SFII, which establishes the real interconnect paths used to implement the target architecture information transfer paths (target architecture external and internal interconnection paths). It is composed of the control codes and software packages that create, regulate, and monitor the interconnect paths needed in the emulating system to implement the user architecture. The monitoring capability of SFII is used as necessary to factor or modify data movement times to determine target architecture performance. SFIIs apply primarily to the representation of standard architectures, although once an SFII has been created it can be contended, with respect to emulation, that it is a standard architecture component.

3.3.2.3.4 Standardized User-designated Hardware Interface (SUHI)

The attachment of user equipments to the emulation facility is not expected to be common in the near term. However, provision must be made for standardizing the attachment of user hardware to forestall the impact of nonstandard methods of attachment on the generation of emulation code. The SUHI is a combination of hardware and software that permits the user hardware to be attached to the emulation system. There may be several distinct SUHI that allow, for example, attachment at the memory interface (through DMA channels), the I/O interface, or as a special processor channel attachment (control register).

3.3.2.4 Performance Measurement and Collection - As final products, RCSDF produces computation results together with performance measures. Performance measurements have two main categories in our context. One category refers to the user designed system (the system that is being emulated), the other refers to the RCSDF itself (the emulation hardware). The performance measures that a user could be interested in are:

1) accuracy of computation results
2) response time and feedback control
3) throughput rate
4) utilization of various emulated resources
5) system bottlenecks
6) software trace
7) instruction trace
8) memory address trace
9) instruction mix description
10) branching (forward and backward branches, number of instructions executed between branches)
11) page swapping frequencies
12) main memory lockout frequency (conflict frequency)
13) register utilization (back referencing, indexing)
14) I/O traffic trace (sensor, peripheral, analog, digital, etc.)
15) total time spent in task entrances, etc.
16) number of task entrances

17) longest time in task

18) time spent in last task entrance

These user performance measures will be sent to the hosting subsystem, where the user-specified data reduction algorithms (tabulation, graphical representation, etc.) are applied.

The performance measures that an RCSDF developer could be interested in are:

1) maximum and minimum number of users in multiprogramming environment
2) average system down time (e.g., MTBF, MTTR)
3) utilization of various RCSDF resources

The facility is responsible for gathering sufficient data to compute the above mentioned performance measures when desired. To fulfill this responsibility the facility needs to have these capabilities:

- provision of hardware tools and their standard checkout procedure
- independent monitoring techniques
- computation packages for various performance measures
- performance data base management

3.3.2.4.1 Hardware Tools Provision and Verification - The facility should provide all the hardware tools and techniques necessary for performance measurement and recording. It should also be able to verify these equipments through the use of standard checkout procedures. Examples of such hardware tools are sensing probes, amplifiers, and data recording media. The sensors are used to extract software data, such as instruction counter contents, op code, memory addresses, procedure names, and I/O types. Another important tool is an interval timer used for generating measurement interrupts. The measurement intervals should be short enough to minimize distortion, but long enough for the measurement data to be computed.

3.3.2.4.2 Independent Monitoring Techniques - The facility should be able to monitor performance without requiring modifications to the user program. It should also be able to install the monitoring devices without detailed knowledge on the operating system of the deployable system. Since the facility is required to emulate many different machines, table-driven operating systems seem to be very desirable.

3.3.2.4.3 Performance Measurement Packages - The performance measures mentioned in the beginning of 3.3.2.4 need to be computed from raw data trapped by the hardware monitoring tools. The facility should provide efficient computation packages for various performance measures, and these computed performance measures should be as concise as possible to reduce the performance data reduction responsibility of the host can be eased.

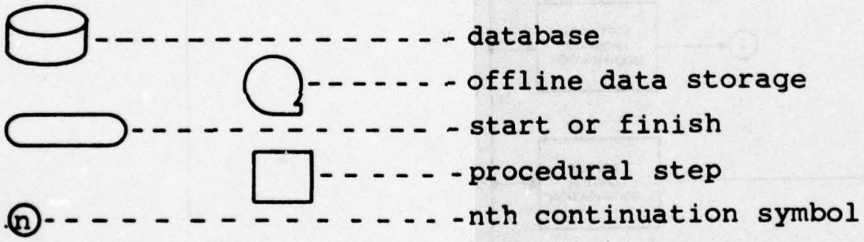
3.3.2.4.4 Performance Data Base Management - The facility should be able to temporarily store the raw data trapped by the monitoring tools until they are processed (computed) by the computation packages. The computed measures will then have to be stored in large secondary storage (disk or tape) before being accessed for data reduction by the hosting subsystem. The facility will have to be able to format the performance measures to the requirements of the host.

3.3.2.5 Data Logging - The facility is responsible for a data log on all data transfers between various facility hardware components (including special user hardware). To achieve this, the facility should be able to record the data transferred at every component interface together with the system clock time at which the data transfer takes place. This data log should be available to the user as an emulation trace whenever the user suspects an error in emulation. A feasible way of providing this data log is to tap all component interface lines into data recording equipment, time multiplexing if necessary. Then the interface transfer signal is used to activate the actual data logging procedures. This technique will allow the data logging to be done without degradation to the emulation system performance. Data transfers between modules residing on the same component are initiated by emulated "hooks" generated by the host. The facility should, however, also be able to recognize these "hooks" and initiate the actual data logging procedure.

3.3.2.6 Test Support - The facility responsibility discussed in this section must assist users to the extent required in the performance of system testing. In order to fulfill this responsibility, the facility requires a staff of personnel with a wide range of skills. Included in these skills are system architecture (resolution of architectural faults discovered during testing), system integration (to provide assistance in resolving interface problems and integration-related software problems), emulation (to resolve problems in generated emulation microcode), performance monitoring (to set up, operate, and resolve problems with performance monitoring equipment), equipment operation and maintenance. Through these provided skills the emulation system is installed, verified, operated, and the performance data collected. The extent of support is dependent on the particular user and the problem he is emulating. However, across the entire user class these skill levels will be required.

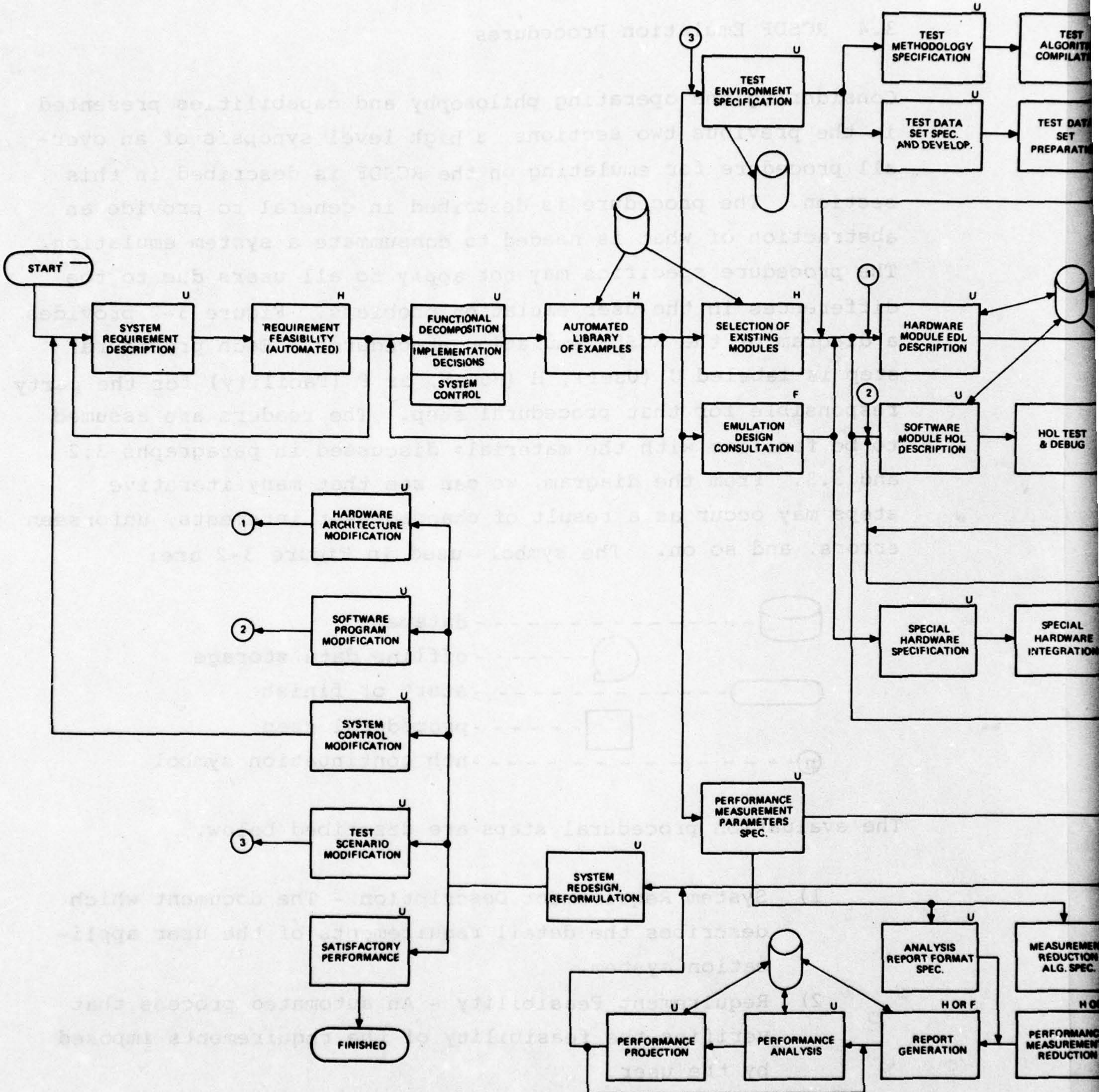
3.4 RCSDF Emulation Procedures

Considering the operating philosophy and capabilities presented in the previous two sections, a high level synopsis of an overall procedure for emulating on the RCSDF is described in this section. The procedure is described in general to provide an abstraction of what is needed to consummate a system emulation. The procedure specifics may not apply to all users due to the differences in the user emulation problems. Figure 3-2 provides a diagram of the RCSDF emulation procedures. Each procedural step is labeled U (User), H (Host), or F (Facility) for the party responsible for that procedural step. The readers are assumed to be familiar with the materials discussed in paragraphs 3.2 and 3.3. From the diagram, we can see that many iterative steps may occur as a result of changed user interests, unforseen errors, and so on. The symbols used in Figure 3-2 are:



The evaluation procedural steps are described below.

1) System Requirement Description - The document which describes the detail requirements of the user application system.
2) Requirement Feasibility - An automated process that verifies the feasibility of the requirements imposed by the user.
3) Functional Decomposition - The decomposition of the application system into functional units or modules.


START
U
SYSTEM REQUIREMENT DESCRIPTION
H
REQUIREMENT FEASIBILITY (AUTOMATED)
U
FUNCTIONAL DECOMPOSITION
IMPLEMENTATION DECISIONS
SYSTEM CONTROL
H
AUTOMATED LIBRARY OF EXAMPLES
3
U
TEST ENVIRONMENT SPECIFICATION
H
SELECTION OF EXISTING MODULES
F
EMULATION DESIGN CONSULTATION
U
TEST METHODOLOGY SPECIFICATION
TEST ALGORITH COMPILATI
U
TEST DATA SET SPEC. AND DEVELOP.
TEST DATA SET PREPARATI
1
U
HARDWARE MODULE EDL DESCRIPTION
2
U
SOFTWARE MODULE HOL DESCRIPTION
HOL TEST & DEBUG
U
SPECIAL HARDWARE SPECIFICATION
SPECIAL HARDWARE INTEGRATION
U
HARDWARE ARCHITECTURE MODIFICATION
1
U
SOFTWARE PROGRAM MODIFICATION
2
SYSTEM CONTROL MODIFICATION
U
TEST SCENARIO MODIFICATION
3
U
SATISFACTORY PERFORMANCE
FINISHED
U
PERFORMANCE MEASUREMENT PARAMETERS SPEC.
U
SYSTEM REDESIGN, REFORMULATION
U
ANALYSIS REPORT FORMAT SPEC.
MEASUREMEN REDUCTION ALG. SPEC.
H OR F
REPORT GENERATION
H O
PERFORMANC MEASUREMEN REDUCTION
U
PERFORMANCE ANALYSIS
U
PERFORMANCE PROJECTION

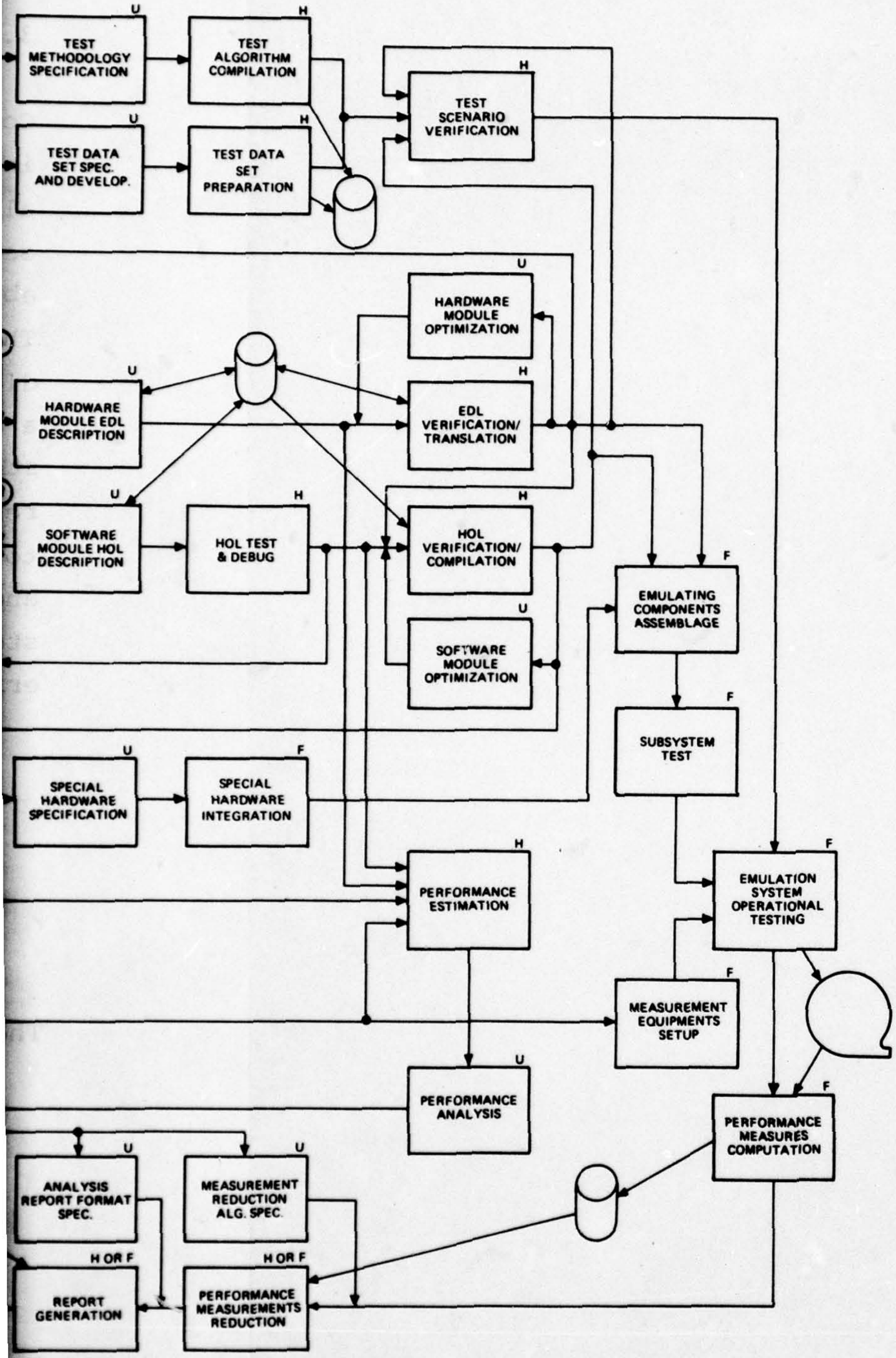

U
TEST METHODOLOGY SPECIFICATION
H
TEST ALGORITHM COMPILATION
H
TEST SCENARIO VERIFICATION
U
TEST DATA SET SPEC. AND DEVELOP.
H
TEST DATA SET PREPARATION
U
HARDWARE MODULE OPTIMIZATION
U
HARDWARE MODULE EDL DESCRIPTION
H
EDL VERIFICATION/ TRANSLATION
U
SOFTWARE MODULE HOL DESCRIPTION
H
HOL TEST & DEBUG
H
HOL VERIFICATION/ COMPILATION
F
EMULATING COMPONENTS ASSEMBLAGE
U
SOFTWARE MODULE OPTIMIZATION
F
SUBSYSTEM TEST
U
SPECIAL HARDWARE SPECIFICATION
F
SPECIAL HARDWARE INTEGRATION
H
PERFORMANCE ESTIMATION
F
EMULATION SYSTEM OPERATIONAL TESTING
F
MEASUREMENT EQUIPMENTS SETUP
U
PERFORMANCE ANALYSIS
F
PERFORMANCE MEASURES COMPUTATION
U
ANALYSIS REPORT FORMAT SPEC.
U
MEASUREMENT REDUCTION ALG. SPEC.
H OR F
REPORT GENERATION
H OR F
PERFORMANCE MEASUREMENTS REDUCTION

Figure 3-2. RCSDF Emulation Procedures

2

4) Implementation Decision - The decision to use either hardware or software implementation for each functional unit.

5) System Control - The description of the control synchronization and interface among identified functional units.

6) Automated Library of Examples - The host library collection of implementation examples; helps the user in finalizing his decomposition process.

7) Test Environment Specification - The document which specifies the scenario and test algorithm under which the emulation test is to be performed.

8) Selection of Existing Modules - Choosing implemented modules from the automated library. (The user should select as many as possible of his modules from among the existing modules available in the library.)

9) Emulation Design Consultation - The expertise and consulting service offered by the facility to improve user emulation design.

10) Performance Measurement Parameters Specification - The document which specifies the performance parameters which the user intends to measure and examine.

11) Test Methodology Specification - The document which specifies the test methods, test sequencing, and post-test data reduction methods for the testing phase.

12) Test Data Set Specification - The document which specifies the test data sets that are used by the test algorithms to cause specified system behaviors.

13) Test Algorithm Compilation - Translation of user-defined test algorithms into facility driving function code and facility driving function sequences.

14) Test Data Set Preparation - Use of the user specified test data set to simulate system inputs for the emulation process.

15) Test Scenario Verification - Verification of the timing between test algorithms and test data sets at the system level.

16) Hardware Module EDL Description - The document which describes each identified module in EDL (Emulation Design Language).

17) Software Module HOL Description - The document which describes each identified software module in HOL (High Order Language).

18) HOL Test and Debug - Debugging of the HOL programs on the host system before emulating on the facility.

19) EDL Verification/Translation - Automated verification of the EDL descriptions of the hardware modules and the translation into facility emulation code.

20) Hardware Module Optimization - Tailoring certain hardware modules into facility emulation code for optimization purposes.

21) HOL Verification/Compilation - Automated verification of the HOL descriptions of the software modules and the compilation into deployable machine code.

22) Software Module Optimization - Tailoring certain software modules into deployable machine code for optimization purpose.

23) Special Hardware Specification - The document which specifies the special hardware attachments to the deployable system including necessary tailoring to meet interface standards.

24) Special Hardware Integration - The integration of user specified hardware into existing facility hardware and the subsequent operational testing.

25) Emulating Components Assemblage - The total integration of all facility modules (both hard and soft) and all user designed modules (both hard and soft).

26) Subsystem Test - Validation of the operability of the separate system components and intercommunication between interconnected components.

27) Emulation System Operational Testing - The integration testing of the entire emulation system, including interface testings.

28) Performance Estimation - Rough estimation of performance for target architecture according to test data set and functional description of the modules.

29) Measurement Equipments Setup - Provision of the performance measurement equipments as required by the user specification.

30) Performance Measurement Computation - Computation of first level performance measurements, e.g., time spent in certain tasks, or frequency of certain instructions.

31) Performance Analysis - Analysis of the performance results either from estimation or from emulation testing.

32) Analysis Report Format Specification - The document which specifies what the analysis report (generated by the host) should look like.

33) Measurement Reduction Algorithm Specification - The document which specifies the reduction algorithms for the performance data obtained during emulation.

34) Performance Measurements Reduction - Reduction of test and performance data according to the user specification and requirement.

35) Report Generation - Generation of performance report according to the user specification.

36) Performance Projection - Projection of measured performance to predict the performance of the target architecture by modification of measured parameters.

37) System Redesign and Reformulation - Redesign of the target system architecture using results from performance analysis.

38) Hardware Architecture Modification - Modification and improvement of unsatisfactory hardware modules.

39) Software Program Modification - Modification and improvement of unsatisfactory software programs.

40) System Control Modification - Improvement of the system control and resource sharing structures.

41) Test Scenario Modification - Modification of the test algorithms and the test data set, if required.

42) Satisfactory Performance - Satisfactory performance of the target system, as demonstrated by the test results.

4. RCSDF Development Plan

This section of the document defines the tasks identified during the Initial RCSDF design study, the results of which are required to meet the overall objectives of the Reconfigurable System Design Facility (RCSDF). To aid in the task selection process, the study team first identified a series of additional concept formulation studies listed in paragraph 4.1 that would enable more definitive objectives to be established for the emulation facility, i.e., the RCSDF portion of the Total System Design Facility (TSDF). This was followed by the establishment of six development paths (relatively independent categories of development effort) documented in paragraph 4.2, for which smaller work and study tasks were independently defined. Finally, the sets of tasks resulting from concept formulation or from the developmental paths were combined into single-task sets, covered in paragraph 4.3, but only after entries which were redundant or ambiguous with respect to RCSDF objectives were removed or clarified, respectively.

4.1 Definitive Study Tasks

The tasks described in this section are intended to support the development of hardware and software operating system specifications for a 1981-1982 RCSDF by providing necessary definitions for procurement of those specifications. The tasks outlined herein will, if pursued, develop source material to be included within procurement documents to define system architecture, control structure, the design language process, and the specification hierarchy to be used.

4.1.1 <u>Emulation System Architecture</u> - The STARAN, QM-1, Data Manipulator Unit, 10^9-bit mass memory, and microprocessor array are system components which can be configured in several or more system architectures. As yet, no architectural definition exists for these components as a system. One will be needed to develop specifications for hardware and software.

4.1.1.1 Task Description - The objective of this task is to prepare preliminary architectural descriptions for the RCSDF, assuming components with the capabilities as represented in the above devices; evaluate these architectures in terms of their use in different classes of user problems; and rank the different architectures in terms of their usefulness for emulating the different problem classes.

4.1.1.2 Examples - A user may visualize the solution to his processing problem in any number of system structure forms. His visualization may or may not be influenced by his knowledge of what exists in the RCSDF (may is more probable than may not). The users visualization could be, for example, any of the following:

Network - A set of processing units, logically equal with one another, loosely coupled, operating autonomously, controlled through mutual arbitration between processing units.

Mainframe - A set of processing capabilities organized as a mainframe design operated, perhaps, sequentially.

Sequential Emulator - A general purpose unit computer employing sequential computation principles.

Parallel Processor - A special purpose parallel processor with identical processing units slaved to a common instruction stream.

Hierarchical System - A general purpose processor controlling special purpose processors that provide unique processing capabilities.

The above visualizations could conceivably be realized as in Figures 4-1 through 4-5 utilizing the planned RCSDF hardware:

Figure 4-1 Network - Each device in the system is loosely coupled to other devices through I/O channels. Devices, where programmable, communicate with each other through high level protocol and regulate their own resources.

Figure 4-2 Mainframe - Each device in the system is tightly coupled with other devices operating in sequential fashion to perform the processing problem.

Figure 4-3 Sequential Emulator - Emulation is centered on the QM-1. The STARAN and other devices support the emulation by providing services, such as data management.

Figure 4-4 Parallel Processor - Emulation is centered on the microprocessor array with the QM-1 acting as the control unit.

Figure 4-5 Hierarchical System - Emulation is centered on the QM-1 microprocessor array combination with the elements of the microprocessor array acting as special purpose controllers.

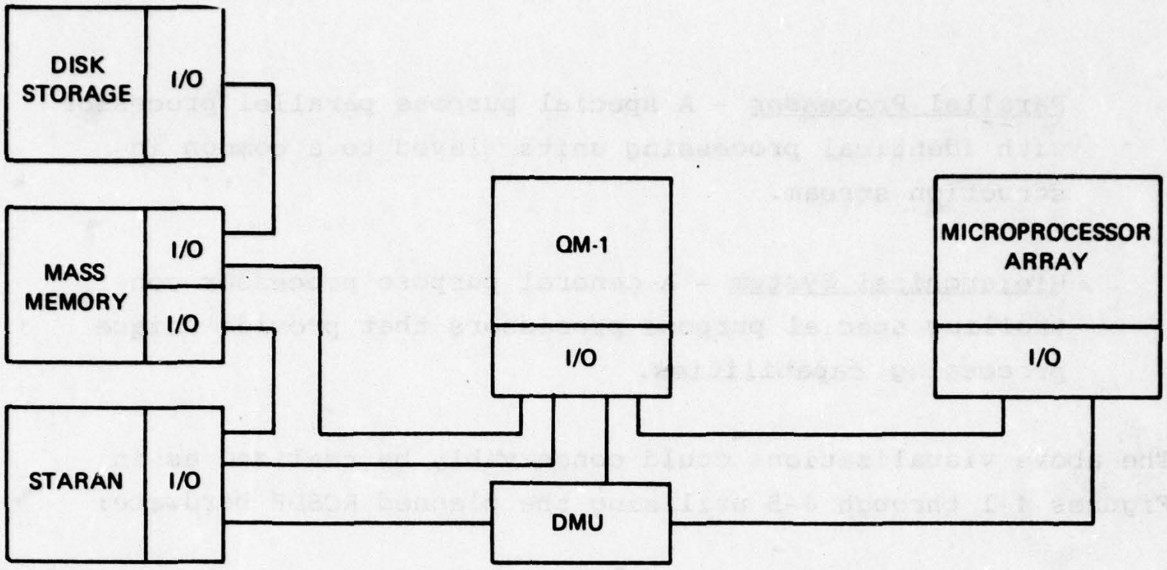


Figure 4-1. Network

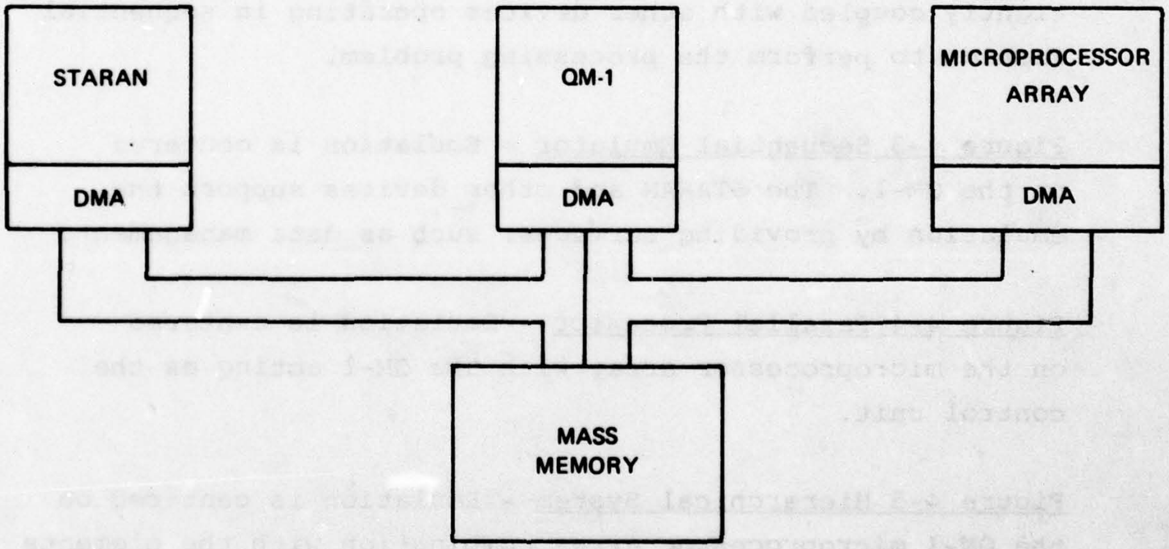


Figure 4-2. Mainframe

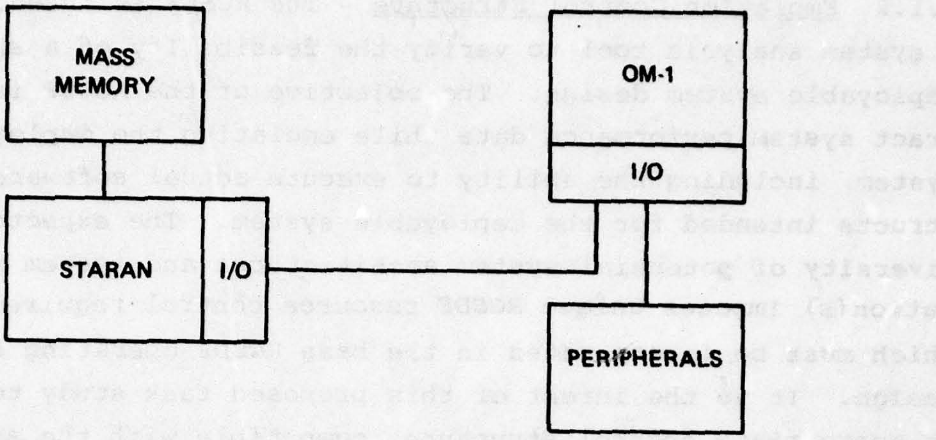


Figure 4-3. Sequential Emulator

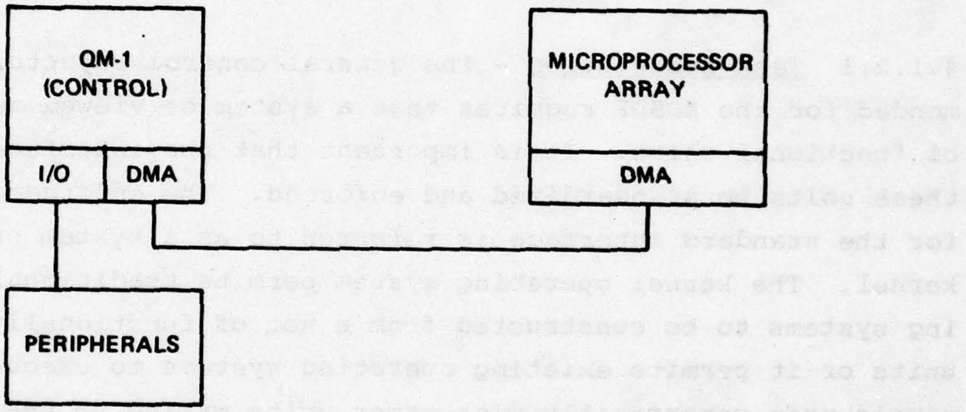


Figure 4-4. Parallel Processor

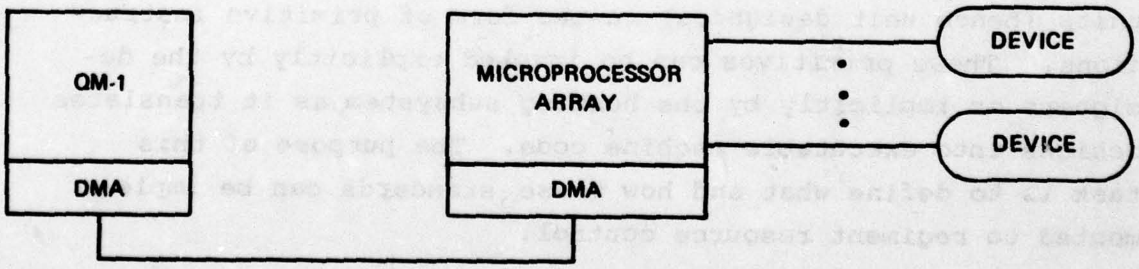


Figure 4-5. Hierarchical System

4.1.2 Emulation Control Structure - The RCSDF is intended to be a system analysis tool to verify the feasibility of a specific deployable system design. The objective of the RCSDF is to extract system performance data while emulating the deployable system, including the ability to execute actual software constructs intended for the deployable system. The expected diversity of potential system architectures and system application(s) imposes unique RCSDF resource control requirements, which must be incorporated in the base RCSDF operating system design. It is the intent of this proposed task study to define an appropriate control structure, compatible with the selected RCSDF architecture, which will satisfy the resource control requirements for a variety of application system architectures.

4.1.2.1 Task Description - The general control structure recommended for the RCSDF requires that a system be viewed as a set of functional units. It is important that the interface of these units be standardized and enforced. The enforcement logic for the standard interface is referred to as a system nucleus or kernel. The kernel operating system permits traditional operating systems to be constructed from a set of functionally complete units or it permits existing operating systems to execute as a single unit concurrently with other units making up the application system.

The kernel permits standard interface requests to be made by the units (hence unit designers) in the form of primitive instructions. These primitives can be invoked explicitly by the designers or implicitly by the hosting subsystem as it translates designs into executable machine code. The purpose of this task is to define what and how these standards can be implemented to regiment resource control.

4.1.2.2 Example of Control Structure Enforcement - The example given here, while superficial, serves to illustrate what is meant by an enforced emulation control structure. Again, assume that access to all RCSDF system resources must be described to a kernel operating system physically executing on the RCSDF hardware. The actual description may come from the hosting subsystem or from units executing on the RCSDF.

If the mainframe architecture (Figure 4-2) is chosen, a kernel supplying identical functions could be implemented in each processing element, i.e., the STARAN, the QM-1, and the microprocessor array. Thus a SHARE MEM instruction, executed on the QM-1 on behalf of a unit executing on the STARAN, would cause the respective kernels to cooperate so that impacted units could access the same mass memory space.

If the network architecture (Figure 4-1) is chosen, the STARAN could be dedicated to the function of virtual memory management with the SHARE MEM primitive the responsibility of only that portion of the system kernel located in the STARAN (i.e., the QM-1 and microprocess array kernels would forward this primitive to the STARAN for subsequent execution and memory sharing enablement).

A slight variation in the latter functional mapping would place the responsibility for all primitives except those dealing with memory management in the QM-1, thus dedicating the microprocessor array for unit emulation. Resource requests (primitives) executed by the units operating in the array would be forwarded to either the STARAN or QM-1, depending upon whether they were a request for memory or other resources, respectively. This would permit effective performance measurements to be recorded by these processing elements.

4.1.3 Emulation Documentation Structure - Specifications for RCSDF hardware and software form a subset of the overall set of documentation required for a user emulation event, an instance of usage of the RCSDF. Initial design studies indicate a need for documentation to coordinate the emulation procedure. Because the use of an emulation facility (i.e., the RCSDF) as an integral part of design is new, an overall documentation structure has not yet been defined.

4.1.3.1 Task Description - The objective for this task is to develop an overall documentation structure for the RCSDF identifying needed documents and their relationship to one another. The task should include the preparation of an overall governing document for the emulation procedure. It should consider that RCSDF operations may eventually image TSDF operations. Insight into the documentation required will aid in evaluating the total system design strategy.

4.1.3.2 Documentation Examples - The following are examples of documents that could conceivably be required to document an emulation in a TSDF, and a subset of which could be required in the RCSDF:

System Requirement Description
Emulation Design Specification
System Description
Test Environment Specification
Hardware Requirements Specification
Performance Measurement Specification
Facility Interface Standards
Performance Measurement Specification
Facility Interface Standards
Performance Analysis Specification
System Integration Specification
System Test Procedure

4.1.4 Requirements/Design Language Procedure

Earlier in the initial RCSDF design study work the process of system design was identified as one of conception and specification (human subsystem), implementation (hosting subsystem), and feasibility testing (reconfigurable subsystem, i.e., the RCSDF). These fundamentally distinct processes are not, however, independent. It is apparent that for the system design process to run its course smoothly, human beings must be able to communicate; automation tools must be developed to translate human operations into functional components; and finally, modular, off-the-shelf (when possible) components must be configured and tested to determine if application requirements can be met in a cost/performance competitive manner. Central to each of these distinct processes is the requirement for unambiguous specifications, i.e., well-defined design languages and usage procedures.

4.1.4.1 Task Description - The purpose for this task is to identify those languages which are required for system specification, design, and development. Specifically, this includes hardware/ software requirement specification and design languages. Common characteristics of each will be identified and a composite or hybrid language defined. The term "hybrid" refers to a standard format or common syntactical base around which several "levels" or language "dialects" can be defined. Each level is intended to bring the top level (requirements) closer to an ultimate implementation. Lower levels are expected to identify the various methods of implementation.

4.1.4.2 <u>Task Example</u> - To illustrate the various language levels which can exist for a specific system "unit" during system design, an example has been taken from the Univac DSD Distributed Processing System Development Laboratory. Selection of the sample unit was done for convenience in producing this document. The function of the unit probably has no application within the RCSDF.

The system from which the unit is taken provides a multi-user program generation capability. The function is defined using three language dialects. The first defines the requirement for the function and hence its role in the system. The second provides functional detail but is still implementation independent. The third describes a primitive operation (block semaphore) with sufficient detail (register transfer level) that it could be implemented in firmware should the data types be carefully defined. For example:

1) <u>Requirement Specification</u>

```
BEGIN Time Slice Scheduler
    Paragraph X.5 Scheduling Philosophy
    Whenever more than one user exists for this system,
    he shall be given access to the LEVEL 2 Operating
    System executing on a 16-bit virtual machine, (VMP).
    The system is expected to permit each user's pro-
    gram(s) to progress at the same rate as other users.
END
```

2) <u>Functional Detail (Structured Narrative)</u>

```
BEGIN Time Slice Scheduler
BEGIN variable:  VMPMAX = number of users in system
WHILE more than one user (VMPMAX>1)
```

```
DO J=1 TO VMPMAX
  Priority VMPj = 7
  Priority for LEVEL 2 of VMPj = 8
END
I = 1
WHILE number of users remain constant
  Priority of VMPi = 7
  Priority of LEVEL 2 of VMPi = 8
  IF I = VMPMAX
  THEN I = 1
  ELSE I = I + 1
  ENDIF
  Priority of VMPi = 5
  Priority of LEVEL 2 of FMPi = 6
  BLOCK ON SEMAPHORE (Clock)
  Release clock unit
ENDWHILE
ENDWHILE
END
END
```

3) Register Transfer Level

(The following description is for the BLOCK ON SEMAPHORE primitive, an extension to the machine repertoire made available by the kernel operating system implemented in Univac DSD's distributed processing Systems Development Laboratory (SDL). The language used is an adaptation of ISP defined by C. G. Bell and A. Newell. The description covers only one function needed by the Time Slice Scheduler defined in the previous "language" examples.)

```
BOS/Block on Semaphore: =
   (SEM ε semaphores owned [AP*]→
   (interrupts [system] ← Illegal Primitive; next cycle);
   next value <SEM> ← value <SEM> +1; next value <SEM>
   >0→ (blocked processes [SEM] ← AP task active <system>
   ←0; dispatch active <system> ←1))
 * AP - Active Process
```

4.1.5 Uniform Emulation Method - A major objective of the RCSDF is to demonstrate the feasibility of the general purpose emulation concept. Essentially this is a many-to-one mapping of one architecture upon another (the RCSDF). Previous emulation efforts have striven for an efficient one-to-one mapping and have not attempted to define a uniform representation structure for many-to-one emulations. If a common representation structure can be defined, then the development of an emulation would be made easier as well as the testing required to validate a specific emulation design.

4.1.5.1 Task Description - The objective of this task is to analyze existing emulation techniques constructed on the QM-1 to determine if common emulation procedures were used to emulate the different machines on the QM-1. The task should examine how other known machines could be represented on the QM-1. This would permit commonality to be detected and projections to be made on the feasibility of a common emulation technique for the RCSDF.

4.1.5.2 Emulation Examples - QM-1 emulations have at least been studied for the computers listed below. At least one of these (S/360) has been implemented, and two others probably are implemented (PDP 11/45, AN/UYK-20).

S/360

PDP 11/45

AN/UYK-20

H-6180

Additional machines that would serve as emulation studies would be:

Univac 1100/80

Burroughs B-1700

Burroughs D-Machine

SKC-200

SEL 32

AN/UYK-7

AN/GYK-12

Interdata 8/32

Rolm/Nova 1664

4.1.6 Emulation Analysis Structure - An objective of the RCSDF is the emulation of deployable systems. The emulation must be capable of permitting actual deployable system components (software and selected special purpose hardware) to execute in an environment where its performance can be measured and analyzed with respect to the application's requirements. An important aspect of the RCSDF configuration is the availability of tools permitting measurement and analysis to take place.

4.1.6.1 Task Description - The availability of tools to permit measurement and analysis to take place requires a carefully defined philosophy for RCSDF structure and resource control. Sperry Univac believes the recommended facility control operating system is compatible with the required philosophy. The objective of this task should be two-fold: first, to describe how the resource control philosophy recommended as the baseline operating system (kernel) can be utilized for performance measurement/analysis; and second, to functionally specify the components (units) required as a part of the RCSDF configuration.

4.1.6.2 Task Example - The example shown in Figure 4-6 assumes that the units executing in the RCSDF are structured in the form of a tree such that "parent" units control use of system resources such as instructions, interrupts, memory, and processors, and that this control is enforced by the kernel operating system.

The units circled represent deployable system functions, which could be a single unit. The operating system units are optional capabilities collected by the RCSDF developers for common application requirements, e.g., file managers and data base managers. The remainder of the units are performance/analysis functions which must be supplied by the RCSDF developers.

The performance measurement unit interfaces with the user to permit measurement requests. For example, if a user requests a count of the number of times a specific system unit (function) executes, the performance measurement unit traps (exercising its control over resources) function calls and records the appropriate data.

If a user does not wish to develop a particular system unit but rather simulate its system response, this fact is described to a scenario generator operating on the host subsystem, fed to the scenario interface unit, and ultimately supplied to the user's system. Whenever this unit would normally execute, the performance measurement unit must trap the dispatch and substitute the simulation data, thus permitting the system to continue.

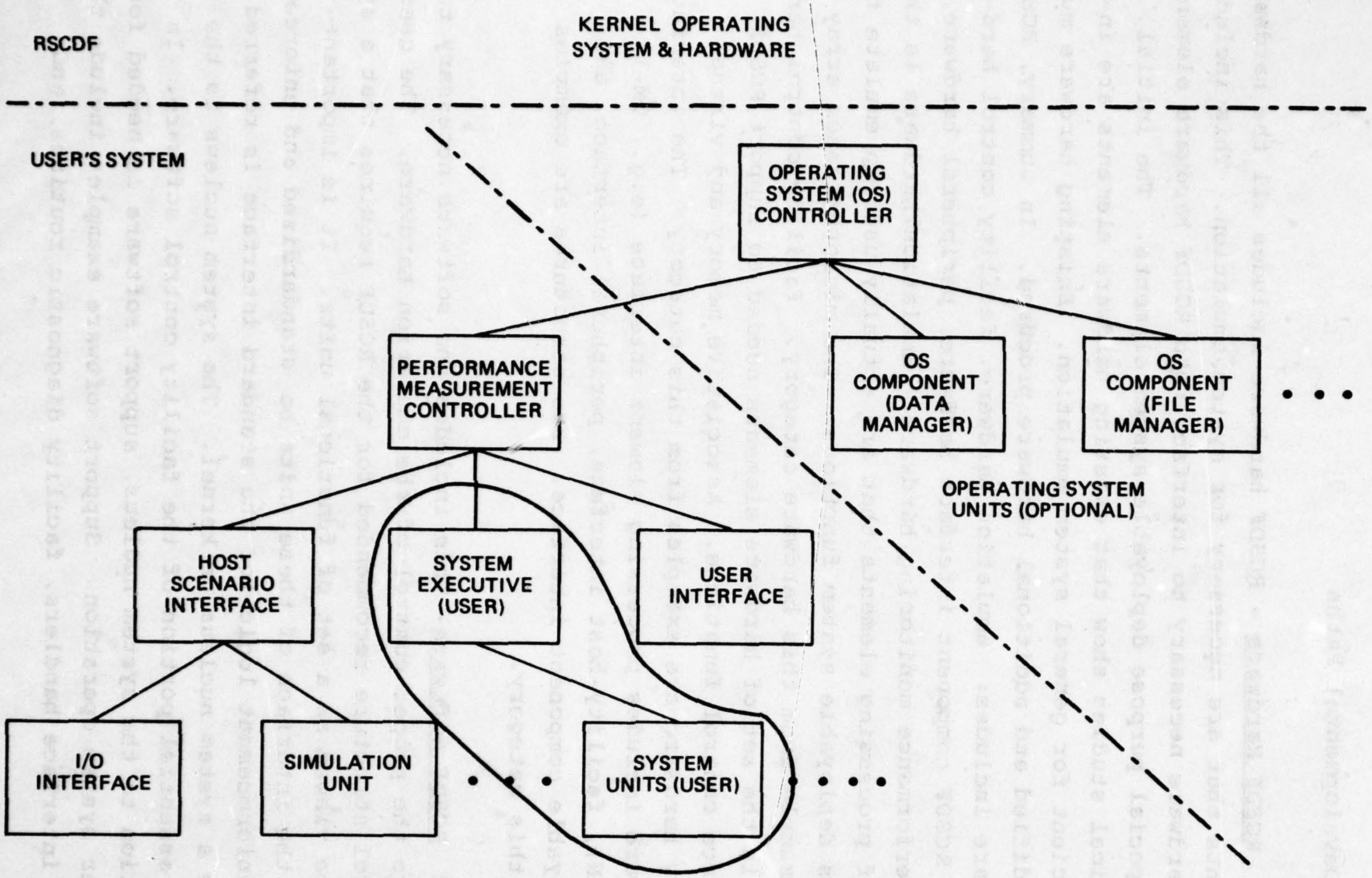

RSCDF
KERNEL OPERATING SYSTEM & HARDWARE
USER'S SYSTEM
OPERATING SYSTEM (OS) CONTROLLER
PERFORMANCE MEASUREMENT CONTROLLER
OS COMPONENT (DATA MANAGER)
OS COMPONENT (FILE MANAGER)
• • •
OPERATING SYSTEM UNITS (OPTIONAL)
HOST SCENARIO INTERFACE
SYSTEM EXECUTIVE (USER)
USER INTERFACE
I/O INTERFACE
SIMULATION UNIT
• • •
SYSTEM UNITS (USER)
• • • •

Figure 4-6. Relationship of Performance Measurement Units

4.2 Developmental Paths

4.2.1 RCSDF Hardware - RCSDF hardware includes all the hardware elements that are necessary for system emulation. This includes the hardware necessary to interface both RCSDF hardware elements and special purpose deployable system elements. The initial technical studies show that existing hardware elements are insufficient for general system emulation. Existing hardware must be modified and additional hardware procured. In summary, RCSDF hardware includes: emulation hardware, facility control hardware, RCSDF component interface hardware, peripheral hardware, and performance monitoring hardware. Emulation hardware is the set of processing elements that are actually used to emulate the user's deployable system functions. The microprocessor array is one example from this hardware category. Facility control hardware is the set of hardware elements needed to support RCSDF resource control functions. Associative memory and virtual memory hardware are examples from this category. The interface hardware includes processing element interface (e.g., QM-1 - STARAN), facility-host interface, peripheral interface, and deployable component interface. Bus structures are examples from this category.

4.2.2 RCSDF Software- This includes the software necessary to insure the proper control of the emulation hardware. The general control structure recommended for the RCSDF requires that a system be viewed as a set of functional units. It is important that the interface of these units be standardized and enforced. The enformcement logic for the standard interface is referred to as a system nucleus or kernel. The system nucleus is the most essential portion of the facility control software. In addition to the system nucleus, support software is needed for proper system operation. Support software examples include the user interface handlers, facility diagnostic routines, and application initialization software. Additional facility software required includes performance measurement software, database management software, and application component simulation software.

4.2.3 Emulation Procedures - Emulation procedures and related descriptions must be carefully developed so that the RCSDF users (system designers) will benefit from the system emulation procedure. This requires that the descriptions establish standard RCSDF utilization procedures, thus enabling a better understanding of the RCSDF system structure. This will permit emulation to be achieved more efficiently.

The user's responsibilities during the emulation phase must be identified. This includes a detailed explanation of the various steps that a user must invoke, starting with the application requirements and analysis and proceeding through functional decomposition. The resulting processes are then described using both hardware and software design languages (as outlined in paragraph 3.4).

A final requirement is a precise description of the RCSDF system architecture so that the user software architecture can be mapped onto the RCSDF hardware components.

4.2.4 Host System Software - The host system is intended to aid the user in the preparation of both selected deployable system elements and the emulation packages necessarily created for the RCSDF. This requires a considerable amount of support software. This includes design language translators and application design database management tools. The latter includes design visibility report generators and the design verification procedures. Host software also includes initialization software, which collects various deployable components (both hardware descriptions and software logic) for inclusion in the RCSDF system.

4.2.5 Design Languages - Design languages are the tools that the designer utilizes to formulate his application system design. During the initial RCSDF design studies contract, six different language types were identified. The language types include Process Design Languages (PDLs), Requirement Specification Languages (RSLs), Emulation Design Languages (EDLs), Scenario Generation Languages (SGLs), Configuration Definition Dialects, and Performance Monitoring Dialects. Only PDLs, and to a lesser degree RSLs, are well represented, with defined languages; the latter four types are still in the conceptual state. To achieve RCSDF objectives a language or a language subset or dialect needs to be selected or specified for each of these types so that the language translators can be developed.

4.2.6 Case Study/Training - To develop system design expertise utilizing the RCSDF, significant case studies should be initiated. The objective of the case studies is to provide the RCSDF designers with sufficient experience to expose deficiencies of the facility and provide insight into corrective activities.

Representative, but nontrivial, applications should be chosen as case studies. For each of the case studies selected, the system functional and performance requirements must be carefully specified. Each functional module or component would then be designed, followed by the actual software (and high risk functional hardware) implementation of the modules using applicable process design languages. Finally,the application system would be emulated.

The experiences obtained from the recommended case studies should be incorporated into a RCSDF user manual, and later used for training purposes.

4.3 Description of RCSDF Development Tasks

This section of the document lists and briefly describes the tasks necessary to implement a general purpose reconfigurable system design facility (an emulation facility) as a part of RADC's total system design methodology.

4.3.1 Emulation System Architecture Study - The STARAN, QM-1, Data Manipulator Unit, 10^9-bit Mass Memory, and microprocessor array are system components which can be configured in several or more system architectures. As yet, no architectural definition exists for these components as a system. One will be needed to develop specifications for hardware and software. This task should prepare preliminary architecture descriptions for the RCSDF, assuming components with the capabilities as represented in the above devices; evaluate these architectures in terms of their use in different classes of user problems; and rank the different architectures in terms of their usefulness for emulating the different problem classes. An evaluation report will be prepared to document the results of this task that will serve as a source document for hardware/software procurement.

4.3.2 Emulation Control Structure Study - The general control structure recommended for the RCSDF requires enforcement logic for the standard interface which is referred to as a system nucleus or kernel. A kernel operating system permits traditional operating systems to be constructed from a set of functionally complete units or it permits existing operating systems to execute as a single unit concurrently with other units making up the application system. The purpose of this task is to define what and how the kernel can be implemented to regiment resource control.

4.3.3 Emulation Documentation Structure Study - The objective of this task is to develop an overall documentation structure for the RCSDF identifying needed documents, their general contents, and their relationship to one another. This will result in the preparation of an overall governing document enabling RCSDF operations to eventually mirror TSDF operations. Insight into the documentation required will aid evaluating the Total System Design strategy.

4.3.4 Requirements/Design Language Procedural Study - The objective of this task is to identify those languages which are required for system specification, design, and development. Specifically this includes hardware/software requirement specification and design languages. Common characteristics of each will be identified so that a composite or hybrid language can be defined.

4.3.5 Uniform Emulation Study - The objective of this task is to determine if a common representation structure can be defined and developed for many-to-one emulations.

Previous emulation efforts have strived for an efficient one-to-one mapping and have not attempted to define such a uniform representation structure. Since a major objective of the RCSDF is to demonstrate the feasibility of the general purpose emulation concept it is essential that a many-to-one mapping of one architecture using the RCSDF architecture be developed.

4.3.6 Emulation Analysis Structure Study - The objective of this task is two-fold. First to describe how the process control philosophy can be utilized for performance measurement/analysis; and second, to specify functionally the components (processes) to permit actual system component measurement and analysis required as a part of the RCSDF configuration.

4.3.7 Additional Hardware Specifications - This is a task resulting from the emulation system architecture study. The objective is to establish specifications for additional hardware that needs to be procured. Examples of possible candidates are the 10^9 mass memory and the microprocessor array. Required functional capabilities of each additional component will be detailed. This includes the component capacity, operational speeds, physical constraints, and desired architectures. Existing components may also need performance improvements. The specifics for areas that need improvement will be described.

4.3.8 Additional Hardware Procurement/Modification and Testing - This is the task of procuring new hardware components and improving existing ones. The task includes the preparation of a bid, the actual building of the hardware,and some supportive software, and the on-site checkout procedures.

4.3.9 Hardware Facility Control Specification - This task follows the emulation control structure study, which identifies a list of desirable executive functions to be implemented in hardware. Hardware logic performing virtual address translation is one example. This task examines how such functions can be implemented in hardware, describes in detail the functional specifications of such hardware units, and establishes the method for integration with software control functions (RCSDF kernel or system nucleus).

4.3.10 Facility Control Hardware Procurement and Testing - The objective of this task is to procure the previously identified hardware functions. Checkout and testing procedures are to be included using either software diagnostics or self-test hardware logic.

4.3.11 Interface Standardization Study - The objective of this task is to standardize interface technologies and methodologies. This includes facility component interfaces, facility-host interface, and peripheral interfaces. Sufficient flexibility shall be defined to enable data path widths and the data transfer rates to be varied for each interface. The interface characteristics (e.g., I/O structures) of each existing or proposed component should also be considered when establishing such standards. This task shall develop an architectural definition for an interconnect system to accommodate foreseeable system expansion.

4.3.12 Communication Protocol Development - This task includes the development of all three levels of communication protocols: the component interface level, the external or I/O level, and the process (software controlled) level. Generally recognized standards shall be used where applicable. In other cases, formats and conventions shall be specified to provide developers with applicable guidelines. This task covers the selection and documentation of protocols identified as RCSDF standards.

4.3.13 Facility Component Interface Specifications - This task follows the emulation system architecture study and the interface standardization study. Interfaces between every pair of facility components are considered. Some such interfaces are not direct connections; instead they are implemented indirectly through another component (e.g., STARAN and the MPA probably interface through the data manipulator, another component). Direct interfaces will be carefully specified, taking expected data flow volume and component I/O structures into consideration. The specifications will follow the standards established in the interface standardization study.

4.3.14 Facility Component Interface Developments - The component interface hardware specified by the task described in paragraph

4.3.13 shall be procured (or developed). This task includes tests for operability of specified interfaces.

4.3.15 Facility-Host Interface Specification and Development - The interface of the facility-to-host interface will be defined. This interface potentially carries the emulation definitions, scenario translations, test data, and performance measurement requests. However, the actual implementation has not been decided; it may be direct connection, dial-up lines, or data supplied by means of magnetic tape. In the possible (but not recommended) case that the facility actually serves as the host, the interface may simply be some storage space in the mass memory.

4.3.16 Peripheral Interface Specification and Development - The interface of facility peripherals shall be specified. This could potentially include enabling peripherals to be connected to a single component or distributed to several components. This task includes the actual development and testing of peripheral interface hardware.

4.3.17 Deployable Component Interface Specification - The recommended RCSDF philosophy permits the user to build his own hardware components and interface them with the facility hardware system. The objective of this task is to specify the interface methodology to be used when interconnecting deployable components into the emulation facility (RCSDF).

4.3.18 Peripheral Hardware Procurement - The objective of this task is to procure the necessary peripheral equipments, such as displays, mass memory, and printers.

4.3.19 Performance Monitoring Hardware Specifications - This task establishes the required facility hardware components for performance monitoring. Identified hardware equipments shall be carefully specified, e.g., the kind of signal detectors required, special timers/counters, and data recording devices.

4.3.20 Performance Monitoring Hardware Procurement - The hardware specified by the task described in paragraph 4.3.19 shall be procured through standard procedures. Procurement includes integrating the equipment with the rest of the facility hardware enabling performance monitoring. Tests of operability are included.

4.3.21 RCSDF Hardware Integration Testing - Individual hardware components should be completely checked out by February 1981. The objective of this task is to integrate all newly procured hardware with the existing system components. Intensive testing for operability of the entire RCSDF hardware system is to be included under this task. The hardware integration testing shall be accomplished using the hardware case studies.

4.3.22 Procurement Procedures Study - This study task establishes the general procedures an application system designer should invoke to procure the hardware and software components for his application. Completion shall result in guidelines for timing the procurement of system modules, both hardware and software.

4.3.23 Responsibility Delineation Description Study - The objective of this study is to identify user's responsibilities during the emulation phase of application system development. Completion of the study will result in preliminary guidelines for total system design utilizing the RCSDF. After completion of case studies, applicable descriptions wili be incorporated into the RCSDF user manual.

4.3.24 RCSDF Structural Description - The intent for the RCSDF is to enable deployable system software and high risk hardware components to be configured and actual execution emulated. This must be achieved without software modification. However, because the RCSDF will be constrained by physical limitations, it will be necessary for the deployable software architecture to be mapped onto the selected RCSDF architecture, i.e., unmodified software processes assigned to hardware components for execution. The objective of this task is to establish guidelines to assist the user in this mapping process.

4.3.25 Application Selections for Case Studies - The objective of this task is to select at least four case studies that will provide the RCSDF staff with sufficient experience to expose deficiencies of the facility and provide insight into corrective action. The case studies chosen should be representative of applications for which the RCSDF will be used.

4.3.26 Requirement Specifications Case Studies - For each of the applications selected by the task described in paragraph 4.3.25, the system functional and performance requirements will be carefully specified (potentially using a requirement specification language). This description shall provide a functional definition of the system which can then be mapped onto functional modules and appropriate control structures, described in paragraph 4.3.27.

4.3.27 Functional Decomposition Case Studies - For each application selected by the task described in paragraph 4.3.25, system functional modules will be identified, isolated, and described. Interfaces between different modules will be defined. Impacts resulting from scheduling and control synchronization shall be specified.

4.3.28 Hardware Implementation Case Studies - For each functional module that has been identified for hardware implementation by the task described in paragraph 4.3.27, an Emulation Description Language (EDL) description shall be prepared. The description shall contain structural and functional aspects of each hardware module.

4.3.29 Software Implementation Case Studies - For each functional module that has been identified for software implementation by the task described in paragraph 4.3.27, High Order Language (HOL) and associated EDL description shall be prepared.

4.3.30 Emulation Driver/Procedures Case Studies - The objective of this task is to prepare the test procedures, test environment, test data set, performance measures required, and data reduction algorithms necessary for each test case emulation exercise.

4.3.31 User Manual Preparation - The experiences obtained from the recommended case studies shall be incorporated into a facility user manual or manuals. Examples shall be given wherever appropriate. Careful documentation of previous tasks shall simplify this task.

4.3.32 Facility Control Software Specification - The RCSDF requires that applications be viewed as a set of interacting, concurrently executing and cooperating processes designed to serve the needs of each other and of the total system. Making resources available to executing processes is the responsibility of the facility control baseline system. The baseline system must permit this construction of processes (satisfying the application requirements) by providing the basic operational control functions (algorithms) required. The implementation of this type of operating system is referred to as a kernel or system nucleus.

The objective of this task is to specify that software be required to perform facility control.

4.3.33 Facility Control Software Procurement and Testing - The procurement phase of the software facility control item shall include the generation and formal approval of program design specifications. Testing shall be performed to the extent that it can assure that all functions defined in the specification are provided in the delivered software packages. Final testing shall be performed during integration testing.

4.3.34 Support Software Specifications - Support software required of the RCSDF consists of utility routines for the loading and testing of software application components. Utilities are also required to alter execution paths, introduce and extract control data, invoke system diagnostics, and provide low level debugging facilities. It is anticipated that many of these utilities shall be provided as software packages delivered with the equipment, thus part of this task shall be to identify those applicable utilities. New and/or modified utility functions compatible with application requirements and the RCSDF operating philosophy must be included in the specifications.

4.3.35 Support Software Procurement and Testing - The procurement phase of the support software procurement and testing item shall include the generation and formal approval of program design specifications. Testing shall be performed to the extent that it can be assured that all functions defined in the specification are provided in the delivered software packages.

4.3.36 Emulation Performance Measurement Software Specification - The primary objective of the RCSDF is to provide an environment capable of emulating deployable systems. It must be capable of permitting actual deployable system components to execute in an environment where its performance can be measured and analyzed with respect to the application's requirements. The structure of the proposed facility control operating system (PX 11868-06) and the level of interface provided to the user's systems suggest that critical performance data must be extracted while executing operating system provided control functions. This specification shall identify the software required to extract performance data and the software required to process this data during and after the emulation process.

4.3.37 Emulation Performance Measurement Procurement and Testing - The procurement phase of the emulation performance measurement item shall include generation and formal approval of program design specifications. Testing shall be performed to the extent that it can be assured that all functions defined in the specification are provided in the delivered software packages.

4.3.38 Support Library Specifications - As application systems are developed by RCSDF users, it is expected that many common hardware and/or software functional units shall be identified and produced. A library shall be developed that will provide an orderly collection of these units and will allow for selection and use by new applications. The methods used to create, update, and maintain this library shall be defined in this specification.

4.3.39 Simulation Software Specification - The objective of this task is to specify the software required to simulate low risk application software modules (processes) during system emulation. Typically, this software translates or reduces data received from the sensors the application's external environment. Since during system emulation only selected data would be used to drive the system, simulation is thought to be most cost effective.

The simulation routines required operate in conjunction with the host subsystem scenario translator to supply test data to the application system components.

4.3.40 Simulation Software Procurement and Testing - The procurement phase of the simulation software item shall include the generation and formal approval of program design specifications. Testing shall be performed to the extent that it can be assured that all functions defined in the specification are provided in the delivered software packages.

4.3.41 RCSDF Software Integration Testing - The objective of this task is to integrate the facility control software, support software, and performance measurement software on the integrated facility hardware. The software integration testing shall be accomplished using the software case studies.

4.3.42 Facility Data Base Management System (DBMS) Specifications - Performance data must be extracted, and simulated data provided during system emulation. Methods are required to process this data as it is generated and entered. It is expected that the amounts of data introduced or extracted and the intervals of activity will vary. Thus, the utilization of hardware components may differ with each application system configuration. The objective of this task is to specify the data base management and lower level mass storage interface functions necessarily installed in the RCSDF. This specification shall include the data base management interface requirements resulting from application and/ or operating system data extraction functions. It shall include the anticipated data rate requirements. Modification to existing data base management capabilities can be specified if compatible with RCSDF requirements.

4.3.43 Facility DBMS Procurement and Testing - The procurement phase of the facility DBMS item shall include the generation and formal approval of program design specifications. Testing shall be performed to the extent that it assures that all functions defined in the specification are provided in the delivered software packages.

4.3.44 Host Data Base Tool Specification - Performance data extracted during system emulation and recorded on the facility DBMS mass storage devices must undergo subsequent data reduction and report generation. The objective of this task is to specify the host software functional requirements and establish the potential types and formats of generated reports. Utilization of available data reduction or report generator software packages shall be specified if applicable. The specification shall include tools capable of providing management visibility relative to the current state of completion of system design. This could be achieved by monitoring the Requirement Specification Language (RSL) as the requirements are translated into Process Design and Emulation Design Languages (PDL and EDL).

4.3.45 Host DB Tools Procurement and Testing - The procurement phase of the host DB tools item shall include the generation and formal approval of program design specifications. Testing shall be performed to the extent that it can be assured that all functions defined in the specification are provided in the delivered software packages.

4.3.46 Static Initialization Software Specification - Deployable system hardware and software components, designed and developed on host facilities, must be collected and bound into a single system or subsystem. The objective of this task is to identify the functional requirements of host software capable of generating a complete system to be loaded into the RCSDF, using RCSDF utility programs. Host system library interfaces, library selection procedure, applicable RCSDF module mapping procedures and structural configuration dependencies shall be defined in this specification.

4.3.47 Static Initialization Software Procurement and Testing - The procurement phase of the static initialization software item shall include the generation and formal approval of program design specifications. Testing shall be performed to the extent that it can be assured that all functions defined in the specification are provided in the delivered software packages.

4.3.48 Language Translators Procurement and Testing - The procurement phase for the language translators shall include the generation and formal approval of program design specifications. Testing shall be performed to the extent that it can be assured that all functions defined in the specification are provided in the delivered software packages.

4.3.49 Host Related Design Language Specifications - The objective of this task is to select or specify three unique language types: the Requirements Specification Language (RSL), the Emulation Design Language (EDL), and the Process Design Language (PDL). Each is identified as a requirement for the total system design concept supporting the RCSDF. An evaluation analysis is required to select or specify each of the three languages.

The RSL is a formal language used to describe the functions and performance characteristics of decomposition units of a system. It does not generate object code, but it is expected that it can be used to correlate unit interface and to verify interconnected data paths. This language is not required to generate systems for execution on the RCSDF, but provides a formal method to describe systems. Further, it is anticipated that it could provide the means for insuring design completeness by correlating functional descriptions (maintained in a data base) with actual program modules generated using the programming languages, i.e., EDLs and PDLs. The EDL is a dialect of a higher order language that shall be used to program the firmware microprogrammed units. The PDL is a higher order language dialect used for programming software units. It is expected that several existing PDLs could be recommended from the analysis associated with this task. The methodology employed to establish a standard Intermediate Exchange Language (IEL) to allow for the collection and execution of units generated by the different dialects of higher order languages should also be specified in this document.

4.3.50 RCSDF Related Design Language Specifications - This set of three user languages is unique to the definition, control, and utilization of the RCSDF. They are expected to operate on a host system. The uniqueness of the dialects suggests that they be defined during the definition phase of the facility control and emulation performance measurement software. The specification of the language capabilities for each dialect is the objective of this task. The Scenario Generation Language (SGL) is a dialect used to formally describe the test environment and testing procedure for a deployable application system. The methods of integrating the output of this language with the outputs of the EDL and PDL languages to produce an input compatible for the static initialization software package must be a part of this task. The configuration definition dialect is used to select and configure units of hardware and software. The performance monitoring dialect is an interactive query language used to formally define critical data introduction and extraction points and to provide for performance data requests.

4.4 RCSDF Work Breakdown Structure

Figure 4-7 shows the RCSDF Work Breakdown Structure (WBS). The format agrees with MIL-STD-881A, 25 April 1975. This standard establishes the criteria governing the preparation of work breakdown structures for use during the acquisition of defense material items. The structure adheres to the definitions associated with electronic systems. The second level definitions applicable to the RCSDF development are:

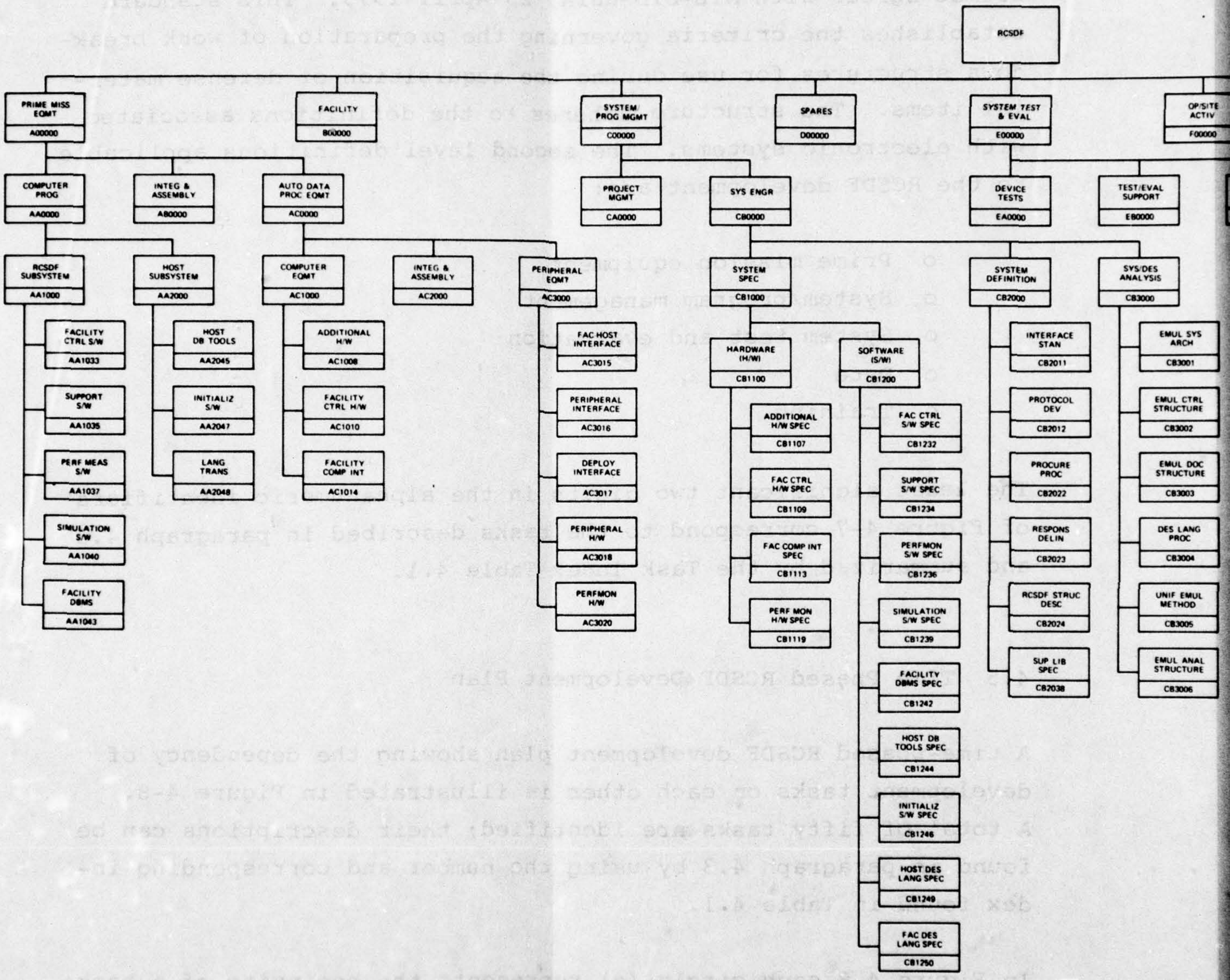
- Prime mission equipment
- System/program management
- System test and evaluation
- Data
- Training

The least significant two digits in the alphanumeric identifiers of Figure 4-7 correspond to the tasks described in paragraph 4.3 and summarized by the Task Index Table 4.1.

4.5 Time Phased RCSDF Development Plan

A time-phased RCSDF development plan showing the dependency of development tasks on each other is illustrated in Figure 4-8. A total of fifty tasks are identified; their descriptions can be found in paragraph 4.3 by using the number and corresponding index found in Table 4.1.

In Figure 4-8 each circle (o) represents the beginning of a task; a triangle (Δ) at the end of the time line represents the completion (C) of a task, or the delivery (D) of the product under procurement; tasks which are prerequisites to the start of other


RCSDF
PRIME MISS EQMT
A00000
FACILITY
B00000
SYSTEM/ PROG MGMT
C00000
SPARES
D00000
SYSTEM TEST & EVAL
E00000
OP/SITE ACTIV
F00000
COMPUTER PROG
AA0000
INTEG & ASSEMBLY
AB0000
AUTO DATA PROC EQMT
AC0000
PROJECT MGMT
CA0000
SYS ENGR
CB0000
DEVICE TESTS
EA0000
TEST/EVAL SUPPORT
EB0000
RCSDF SUBSYSTEM
AA1000
HOST SUBSYSTEM
AA2000
COMPUTER EQMT
AC1000
INTEG & ASSEMBLY
AC2000
PERIPHERAL EQMT
AC3000
SYSTEM SPEC
CB1000
SYSTEM DEFINITION
CB2000
SYS/DES ANALYSIS
CB3000
FACILITY CTRL S/W
AA1033
SUPPORT S/W
AA1035
PERF MEAS S/W
AA1037
SIMULATION S/W
AA1040
FACILITY DBMS
AA1043
HOST DB TOOLS
AA2045
INITIALIZ S/W
AA2047
LANG TRANS
AA2048
ADDITIONAL H/W
AC1008
FACILITY CTRL H/W
AC1010
FACILITY COMP INT
AC1014
FAC HOST INTERFACE
AC3015
PERIPHERAL INTERFACE
AC3016
DEPLOY INTERFACE
AC3017
PERIPHERAL H/W
AC3018
PERFMON H/W
AC3020
HARDWARE (H/W)
CB1100
SOFTWARE (S/W)
CB1200
ADDITIONAL H/W SPEC
CB1107
FAC CTRL H/W SPEC
CB1109
FAC COMP INT SPEC
CB1113
PERF MON H/W SPEC
CB1119
FAC CTRL S/W SPEC
CB1232
SUPPORT S/W SPEC
CB1234
PERFMON S/W SPEC
CB1236
SIMULATION S/W SPEC
CB1239
FACILITY DBMS SPEC
CB1242
HOST DB TOOLS SPEC
CB1244
INITIALIZ S/W SPEC
CB1246
HOST DES LANG SPEC
CB1249
FAC DES LANG SPEC
CB1250
INTERFACE STAN
CB2011
PROTOCOL DEV
CB2012
PROCURE PROC
CB2022
RESPONS DELIN
CB2023
RCSDF STRUC DESC
CB2024
SUP LIB SPEC
CB2038
EMUL SYS ARCH
CB3001
EMUL CTRL STRUCTURE
CB3002
EMUL DOC STRUCTURE
CB3003
DES LANG PROC
CB3004
UNIF EMUL METHOD
CB3005
EMUL ANAL STRUCTURE
CB3006

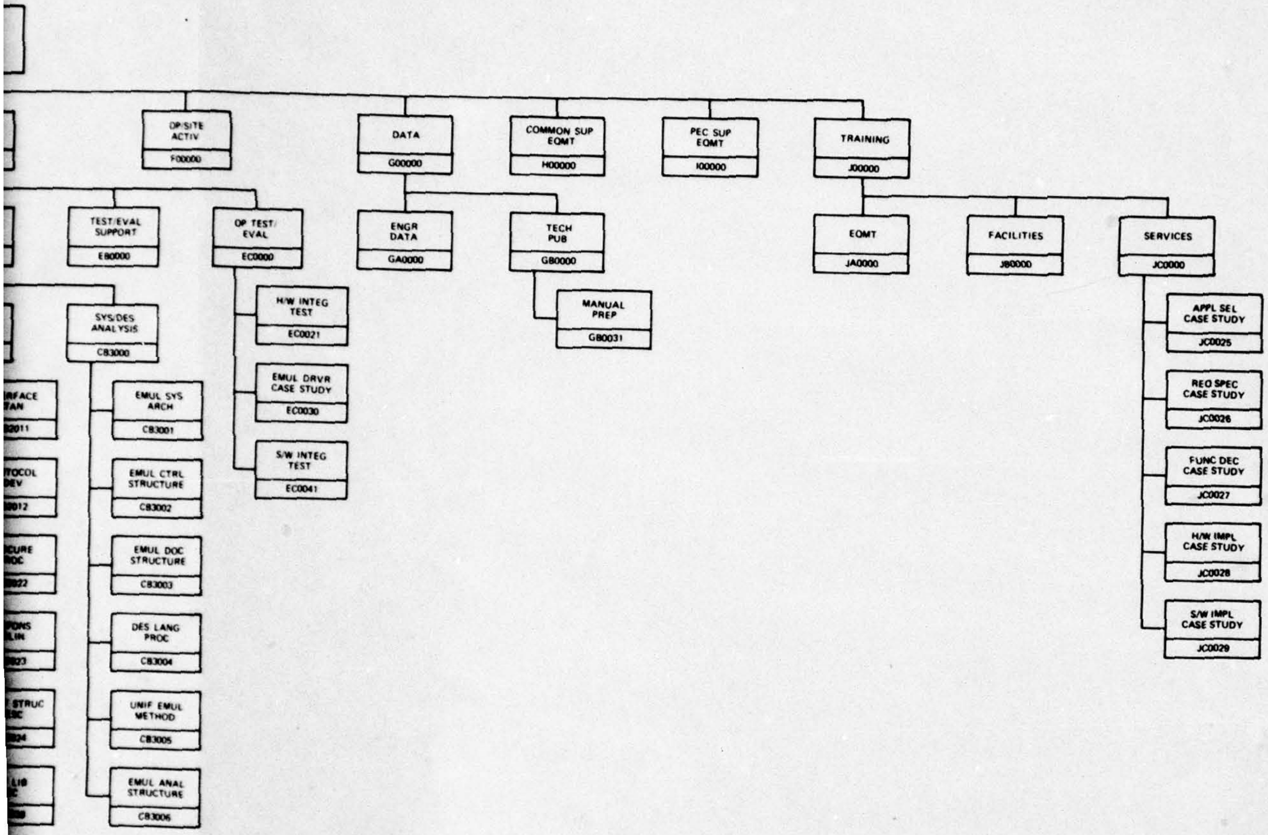


Figure 4-7. Work Breakdown Structure

1. Emulation system architecture
2. Emulation control structure
3. Emulation documentation structure
4. Requirements/design language procedure
5. Uniform emulation method
6. Emulation analysis
7. Additional hardware specifications
8. Additional hardware procurement and testing
9. Facility control hardware specification
10. Facility control hardware procurement and testing
11. Interface standardization study
12. Communication protocol development
13. Facility component interface specifications
14. Facility component interface development
15. Facility-host interface specification and development
16. Peripheral interface specification and development
17. Deployable component interface specification
18. Peripheral hardware procurement
19. Performance monitoring hardware specification
20. Performance monitoring hardware procurement
21. RCSDF hardware integration testing
22. Procurement procedures study
23. Responsibility delineation description study
24. RCSDF structural description study
25. Application selections for case studies
26. Requirement specifications case studies
27. Functional decomposition case studies
28. Hardware implementation case studies
29. Software implementation case studies
30. Emulation driver case studies
31. User manual preparation
32. Facility control software specifications
33. Facility control software procurement and testing

Table 4-1 - Task Index

34. Support software specification

35. Support software procurement and testing

36. Emulation performance measurement software specification

37. Emulation performance measurement software procurement and testing

38. Support library specification

39. Simulation software specification

40. Simulation software procurement and testing

41. RCSDF software integration testing

42. Facility DBMS specification

43. Facility DBMS procurement and testing

44. Host DB tools specification

45. Host DB tools procurement and testing

46. Static initialization software specification

47. Static initialization software procurement and testing

48. Language translators procurement and testing

49. Host related design language specification

50. RCSDF related design language specifications

Table 4-1 (cont'd)

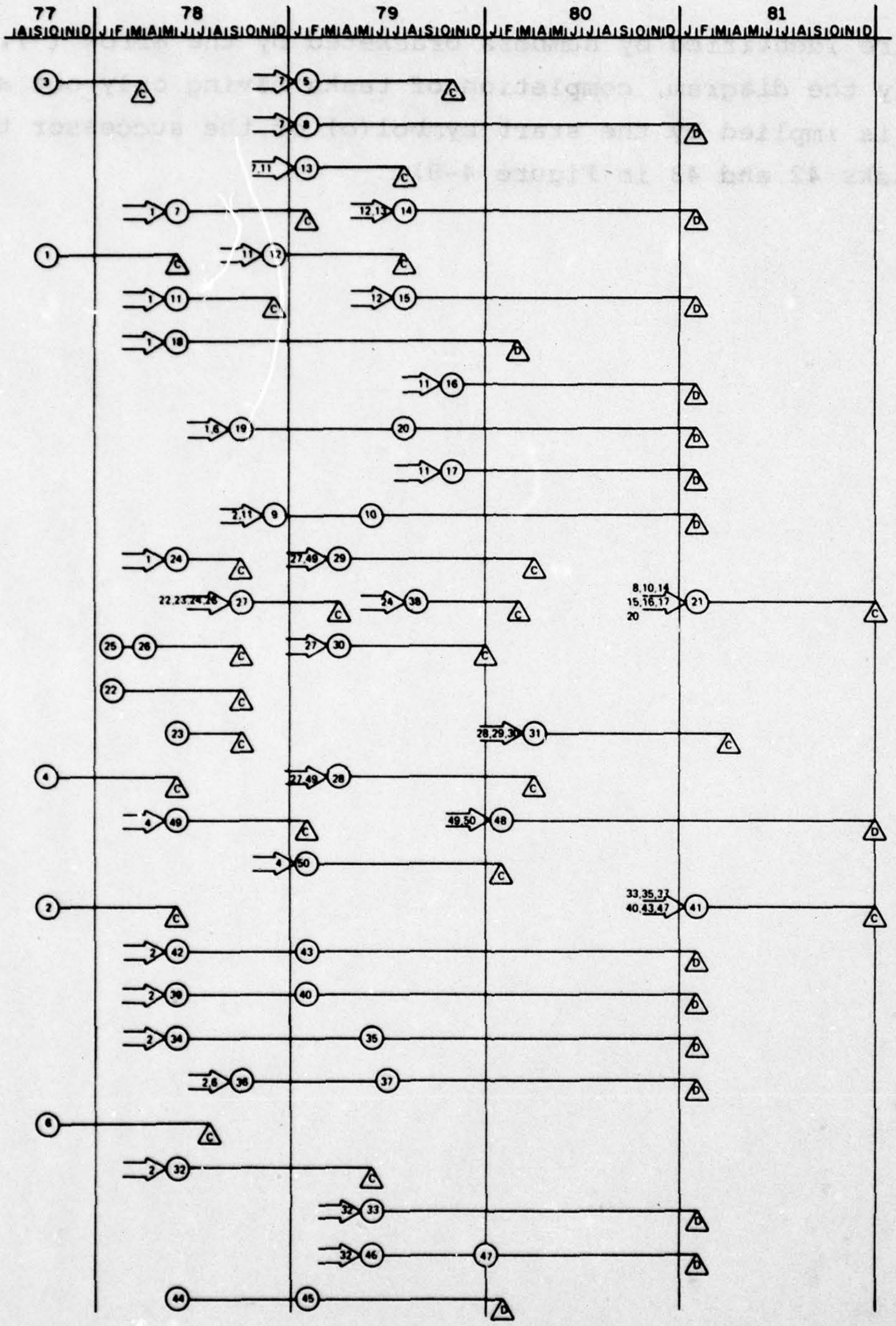


Figure 4-8. Time Phased Plan

tasks are identified by numbers bracketed by the arrow (→). To simplify the diagram, completion of tasks having only one successor is implied by the start symbol(o) of the successor task. (See tasks 42 and 43 in Figure 4-8).

5. RCSDF Baseline Studies

Technical baseline studies in performance measurement, processor communication techniques/protocol, microprocessor networks, operating systems, microprogramming, distributed systems organization, and design languages are summarized in this section. Complete baseline study documentation is contained in Volume III, Initial RCSDF Design Study, Source Documentation.

5.1 Performance Measurement Technical Baseline

Performance measurement has two distinctive phases: monitoring and data presentation. In the monitoring phase, essential performance data are collected from the system whose performance is to be measured. The general consensus is to store these data in a data base and then process them offline in the data presentation phase. In this phase, the vast volume of performance data is digested to produce an easily readable output for the performance analyst.

5.1.1 Monitoring Techniques - In order to measure the performance of a computer system, a great deal of information has to be collected from the system on a regular basis. This data collection is termed monitoring. Computer systems can be monitored by 1) hardware, 2) software, or 3) hybrid techniques.

Hardware monitoring techniques generally examine the monitored system using an independent set of hardware instruments which can sense, decode, count, and record the signals generated by the monitored system. Typically, only effects can be observed. The main advantage of this method is that it does not interfere with the actual operations of the monitored system.

With the software techniques, the system is usually examined by software programs that are embedded in the normal program flow of the monitored system. These software data gathering programs selectively monitor given events at the macro level. The major disadvantage of software monitoring is that each of the measurement programs uses a certain amount of the monitored system resources, hence the performance measured will be somewhat corrupted. The major advantages of this technique are its flexibility and its ability to locate data that cannot be sensed by hardware probes.

The hybrid monitoring method uses a combination of both hardware and software approaches. It employs a physical set of hardware components that is activated by the software of the monitored system. This arrangement potentially offers both measurement flexibility and minimized overhead. Since it is most practical, most measurement systems are this type.

5.1.1.1 Hardware Monitoring Techniques - There are four functional stages of a hardware monitor: signal detection and amplification, concentration and selection, timing and counting, and output devices.

5.1.1.1.1 Signal Detection and Amplification - The first stage of a hardware monitor is the collection of system signals that are needed for the performance measurements. System state signals are observed by high impedance probes that are attached directly to the monitored system circuitry. These signals will then be converted and amplified.

5.1.1.1.2 Concentration and Selection - In many hardware monitoring systems, concentrators are used to reduce the number of lines leading from the sensing probes to the monitoring devices. After the concentrator stage, there is usually a selector stage. Logic elements are required to set up the conditions under which certain signals will be sampled. In the simple cases, only NAND or NOR gates will be used. More advanced monitors will include logic elements such as data comparators, sequencers, event distribution analyzers, random access memories, and associative memories. The comparators may either be set by panel switches or be programmable if the monitoring system contains a separate small computer.

5.1.1.1.3 Timing and Counting - After the required signals have been selected or the right conditions have been met, certain counters will have to be activated to provide a summary of the event counts and durations. A clock must be used with the hardware monitor to provide the timing.

5.1.1.1.4 Output Devices - The counter and timing results, together with certain sensed data values (such as memory addresses), will have to be output to the analyst. The most common form of output device is the magnetic tape recording unit. It records the required performance data values. From this raw data, performance measures that are required by the user will be computed offline at a later time.

Hardware monitoring techniques can be utilized effectively in measurements such as CPU utilization, and I/O channel activities. In some other measurement of interest, they may not be very efficient, and software or hybrid techniques will have to be used.

5.1.1.2 Software Monitoring Techniques - Although hardware monitoring techniques are very flexible and can be used to measure most electrical signals, they cannot be used to measure signals inside a component. This is particularly true with LSI chips, where it is impossible to get at some of the local signals (e.g., the ALU inputs), and hardware monitoring thus becomes very difficult. Software techniques will be needed to gate the necessary information to the external pins of LSI chip, and performance monitoring can then be carried on.

Information can be gathered in a software monitoring environment either by the interrupt method or the sampling method. For interrupt methods, certain traps in the user programs will be recognized. Control will then be transferred to corresponding monitoring routines, where necessary measurement information associated with the particular type of trap will be recorded. For sampling methods, the transference of control to monitoring routines occurs at a regular interval determined by a user specified parameter, the sampling rate. The set of data collected is usually referred to as the system's snapshot. The sampling method has the advantage of generally requiring less system overhead than the interrupt method. Since the interrupt method is event-driven, the user can key on a certain event(s) which may occur at random intervals. The amount of data collected using the interrupt method will thus be minimal. To obtain a comparable amount of useful information with the sampling method, a higher sampling rate is required. As a result, there will be more data collected, taking more time for collecting and processing. Usually the sampling method is used in situations where only statistical information is required (e.g., instruction mix and percentage of CPU busy time). In such situations, a high sampling rate is not needed. On the other hand, interrupt method is normally used in situations where a program trace is required, or where particular events are of interest, e.g., the number of times and length of time spent in different tasks.

Most operating systems available on commercial large scale computer systems provide some sort of measurement and evaluation routines, e.g., UNIVAC 1108's Software Instrumentation Package, IBM S/360's OS/VS, and GECOS on HIS 6000. There are also other vendors who produce standard measurement packages, e.g., Boole and Babbage's Configuration Utilization Evaluator and Problem Program Evaluator, and Tesdata's SUPERMON. These software measurement methods aim primarily at finding the frequency distributions of execution addresses, thus determining which routines have the greatest need for optimization. They can also be used to determine and improve job throughput in the system by recording various equipment utilizations and queue sizes.

5.1.1.3 Hybrid Monitoring Techniques - Performance monitoring has a natural tendency to integrate the pure hardware and the pure software techniques to produce feasible hybrid monitoring methods. A hybrid monitor may be using software methods to gather the data and passing them to external hardware (like another minicomputer). The software traps may be embedded in the host software streams or may be generated as intermachine interrupts coming from the monitoring minicomputer. Another possible hybrid monitor may use hardware probes to collect performance data and software inside the monitoring minicomputer to do the selection and counting processes. However, the former method seems to be more practical, because the latter method is still incapable of capturing signals within a component.

5.1.2 Data Presentation Techniques - After the performance data have been collected by the monitoring system, they need to be processed before an acceptable form can be presented to the analyst. The output from the monitoring system will contain the data required to calculate the performance measures. The first step in data presentation is the calculation of the actual performance parameters. The next step is the data reduction procedure. The actual reduction method has to be specified by the analyst. It may be plotting a separate histogram, or it may be finding average performance values. Data presentation will reduce the enormous amount of data available to just the data of interest to the analyst and present it to him in an easily understood format.

The performance measurement system has to provide the computation packages required to do the performance parameter calculations. If the sampling rate from the monitoring system is not too fast, these computations may be done online, thus eliminating some temporary storage problems (for outputs from the monitoring system). However, they can normally be expected to be done offline.

The three main classes of data reduction techniques are discussed below.

5.1.2.1 Graphic - Graphic results are easy to understand at a glance, but the precision of the measurements tends to be lost. Examples of graphic presentation are charts, point plots, point-to-point curves, and histograms.

5.1.2.2 Tabular - Using tables to present performance data preserves measurement precision, but readability is diminished.

5.1.2.3 Statistical - Providing statistical measures, such as mean, standard deviation, median, expected value, correlation coefficient, and weighted average, can give the analyst greater insight into the correlation of his performance data.

The analyst should be able to specify any one or several of these data reduction techniques. Because of the large volume of data and computations involved, these techniques are usually done offline. The data will be stored in a data base on a secondary storage subsystem, such as disk or tape. It is also common in performance measurement systems to allow the analyst to make interactive queries (using a high level query language) on the performance data base. This allows the analyst to focus on the unsatisfactory areas quickly.

5.1.3 RCSDF Performance Measurement Requirements and Recommendations - The performance measurement techniques described in earlier paragraphs apply to generalized digital system measurements. Some important points that make the RCSDF performance measurement distinct from other systems are:

- RCSDF will be one of the few systems in which performance is measured for a system that is being emulated (or simulated) on another system.

- RCSDF will be the only known system that allows generalized emulation, i.e., is capable of emulating a wide variety of architectures.

- Some RCSDF hardware components (such as the microprocessor array and the data manipulator) use LSI packaging, and it will be impossible to use hardware probes to get at some of the internal signals.

- The software written for the emulated hardware system is written in HOL (High Order Language) and requires a compilation process. Instruction timing information can be inserted at the code generation phase without much difficulty. This favors the use of an emulated clock(s) for performance timing.

- The general responsibilities of the hosting subsystem are enormous; however, the facility has little processing to do.

- Data logging on all data transfers between all emulated modules is a desired feature.

Because of these distinctive properties, there are several special recommendations for measuring performance on the RCSDF:

- Software (or firmware, since microcode can be used) and hybrid monitoring techniques will be necessary to obtain most performance data.

- Hardware probes may be useful to monitor performance data at the component levels, e.g., channel utilization and CPU utilization.

- An emulated clock system should be used to monitor the timing of the emulated system.

- The instruction time associated with each instruction may be calculated at the code generation phase and carried as a field in the instruction.

- To provide data logging between all emulated modules, hardware probes can be used to tap all RCSDF facility component interface lines into the monitoring equipment, using the interface transfer signal to activate the actual data logging procedure. Data transfers between emulated modules residing on the same component are initiated by emulated "hooks" generated by the hosting subsystem. The facility should be able to recognize these "hooks" and initiate the actual data logging procedure using software monitoring techniques.

- Since the hosting subsystem has the relatively heavier load, the performance measures computation should be done in the facility.

The recommendations discussed above should be regarded as supplementary to many of the common and relatively well received techniques discussed in paragraphs 5.1.1 and 5.1.2. For example, all of the data presentation techniques apply to RCSDF also.

Although the area of performance measurement has been continually explored in recent years, a generalized emulation facility is a new concept. This generalized emulation concept creates a great challenge to existing performance measurement techniques. New performance measurement ideas will have to be developed to satisfy the RCSDF requirements.

5.2 Processor Communication Techniques/Protocol Technical Baseline

In the RCSDF emulation concept it is felt that in order to emulate a wide range of system architectures and to pursue architectural research, it is desirable to develop techniques to utilize and control a multiple machine emulating system. In order for the multiple machine emulating system to operate efficiently methods must be conceived and utilized to allow information to be transferred and shared between machines efficiently with a minimum of logical aberrations and interference with overall emulating system operations.

Specific RCSDF requirements in this area have as yet to be clearly specified. Only general needs are known due to the formative nature of the RCSDF.

Topology and Interconnect Implementation - The most effective RCSDF topology to implement is a star topology, one in which every device can communicate with every other device. This kind of topology can be implemented in several ways. A large centralized switch could provide direct paths between any two devices, for example. A ring bus could also be used, but performance would be radically different because the ring bus capacity would be shared between devices. Both implementations when viewed from a process level can provide a flexible topology.

Protocol Hierarchy - Any two devices that communicate with one another use a protocol, In a distributed RCSDF, there appears to be a need for two general kinds of communication etiquette: that which governs the transmission of information between devices without specific regard to content, and that which processes the contents of the transmitted message without regard to the details of moving the information from source to destination in the system. This leads to a protocol hierarchy, one in which the latter kind of etiquette is based upon the former.

Error Control - The processor communications and protocol technique must provide for error control to assure reliable operations. Retransmission is a more desirable technique than error correction for error control. However, retransmission affects performance, performance analysis, and performance monitoring since every retransmission may have to be accounted for in recreating system behavior for extrapolating system performance.

Processor Communications Logging - Essentially, this function records all information moved using the processor communications method and also records scenario particulars about the communication. Such logging assists in the checkout of the emulation, verifying the processes are communicating properly. It is also valuable for recreating system behavior.

Processor communications techniques and protocol will be examined as three distinct system structure types: intercomputer, network, and array. Each of these will be described in the following paragraphs.

5.2.1 Intercomputer Processor Communication - This is the interconnection of colocated machines primarily with I/O channels but can include experimental cases of memory sharing and interprocessor technique.

5.2.1.1 I/O Channel Communications - Intercomputer methods for processor intercommunications are dominantly I/O channel interconnections. Data transfer is usually accomplished on a word or byte basis with request/acknowledge signals causing the transfer.

Topologically, the I/O channel intercomputer connection support of point-to-point connections is limited only by the number of channels available as standard hardware. This limit can often be extended through multiplexers attached to a channel. Star topology in which every computer element directly connected to every other computer element, and substar topology which is a partially connected star, can be implemented with this I/O channel connection scheme. Ring topology can also be implemented with the I/O channel interconnect by providing software to interpret and control the passage of information arov^d the ring.

There are three distinct levels of protocol associated with the intercomputer connection: the interface level convention, the control structure used to regulate I/O buffer transfers, and the process that provides interpretation of the I/O buffer contents. The first level of protocol is specified by the interface standard used in the computer's design. Users have no options to alter specific interface functions or electrical interface characteristics at this level, but may have the option to select from a limited set of alternative interfaces. This level of protocol usually does not provide for the detection of transmission errors.

The second level of protocol is implemented in the design of the computer's I/O control structure. Here the user has some flexibility because the set of options and alternatives is greater and sequences of operation can be partially implemented with software. This level of protocol is usually moderately difficult to implement provided the first level of protocol is successfully accomplished. Sophisticated I/O handlers are usually required on both ends of the communications path. Some of the items included in this second level of protocol include:

- o Size limitations on I/O buffers
- o Buffer format and chain address conventions
- o Interrupt and channel monitor conventions
- o I/O transfer modes
- o Multiplexing

The third level of protocol (message structure/interpretation) is usually completely software. This level constructs, manipulates, sequences, and interprets information to be exchanged between processes. Here the user has the most flexibility. At this level, the greatest possibility for transference between systems occurs because of the decoupling of the protocol process from hardware. General techniques that can be used at this level of protocol are discussed below.

Mailboxing - With this technique processes communicate through pre-assigned memory locations known as mailboxes. Processes check their mail boxes periodically for information that indicates a request for communication. This technique is very practical for systems which contain many processes communicating at different rates.

Direct Registration - When this technique is used the name of the process to be communicated with is located within the information transferred, and the process is immediately placed into the host task queue to be serviced.

Listing - Using this technique process communications are put into a list which is scanned periodically as a task in the task queue. Process communications are assigned priorities, and processes are activated on the basis of that priority.

5.2.1.2 Interprocessor Communcations - Interprocessor communications differ from I/O channel communications in that they occur directly between processor system elements; information is exchanged as a direct action of the processor, not the I/O controller or as an I/O control function. Interrupts associated with the exchange of information may be handled within the processing elements as a part of the normal processor interrupt structure. Protocols in the interprocessor communications are similar to those used with I/O communications, both in the levels of protocol used and the functions performed. The mechanics of implementation may vary due to different machines used.

One example of interprocessor communications hardware is the interprocessor buffer of the Univac EPIC (Experimental Processor with Interprocessor Communications). The PDP-11 UNIBUS window is another.

5.2.1.3 Memory Connected Intercomputing - Interprocess communication through memory has been a fundamental technique in multiprocessing for many years, especially for distributed processing systems that are physically colocated. With this type of communication the source of information is a processor's memory, the destination another memory cell. Multiprocessor architecture mechanizes this concept by sharing memory among processors through a parallel interconnect. Distributed systems can employ a shared bus structure. When this technique is used, the system processors are memory-connected. Protocols used in these situations can be derived from both multiprocessing and distributed system techniques. Dijkstra's semaphore technique, for example, supports the access regulation of shared memory segments which are used for the exchange of information.[3] In order to support this type of intercomputing technique, the communicating processors must have a DMA (Direct Memory Access) feature.

With this type of system connection, protocol at the link is specified by the memory interface specification, protocol at the next highest level by the design of the processor addressing convention, and protocol at the message interpretation level by how the software treats the message data. With memory sharing, the highest level of protocol can employ multiprocessing techniques such as semaphores or test and set capabilities to synchronize accesses and sequences.

Examples of this class of interconnections are the Sperry Univac Memory Multiplex Data Link (MMDL), the Air Force Distributed Processor/Memory (DP/M) bus, and the PDP-11 Unibus Link.

5.2.2 Networks - Computer networks have become a significant system structure during the last decade. The term "network," as applied to computer system structure, has generally been

[4] E. W. Dijkstra, "Cooperating Sequential Processing," in Programming Languages, Genuys, ed. (1968), pp. 43-110.

referred to as a common-carrier based system design. In a more general sense, however, it also means a collection of computers operating in network fashion. This may mean that each computer has an operating system and that each computer communicates with other computers using the same conventions or protocol established for this particular computer collection. The three types of networks discussed in this section are categorized on the basis of the communications technique used to support information movement: common-carrier based network, dedicated bus network, and parallel interconnected network.

5.2.2.1 Common-Carrier Based Networks - The use of a common-carrier network to connect for computer systems has been intrinsic to the development of communications system structures. Communications networks can be classified as one of three types:

RAN (Remote Access Network) - Terminals feed through communications lines to a central host where processing occurs.

VAN (Value Added Network) - A network of autonomous hosts whose operating capability is enhanced through the addition of a communications subnetwork and protocols.

MON (Mission Oriented Network) - A network technologically equivalent to the VAN type, but under control of a single administration. This implies dedication of resources toward prescribed mission functions.

IBM's SNA (Systems Network Architecture) is an example of a RAN network. The VAN type of network is exemplified by the ARPANET. ARPANET employs the IMP (Interface Message Processor) subnet to provide the communications capabilities needed. Routing, error control, and prescribed interfaces with the host machine were built into this subnet. A separate IMP-IMP protocol exists for reliable transmission of packets from IMP to IMP. Also included are higher level protocols such as TELNET, remote job entry, and file transfer protocol.

The MON type of network, based upon common carrier communications is less prevalent. The philosophy of the MON network is that the administration of the entire network as a whole allows priorities and dedication of network resources to be performed, thus making the network more efficient for performing mission functions. The MON philosophy seems to be more aligned with military application networks that are implemented with dedicated bus and parallel interconnect structures.

There are three general communications techniques used in networks: circuit switching, message switching, and packet switching.

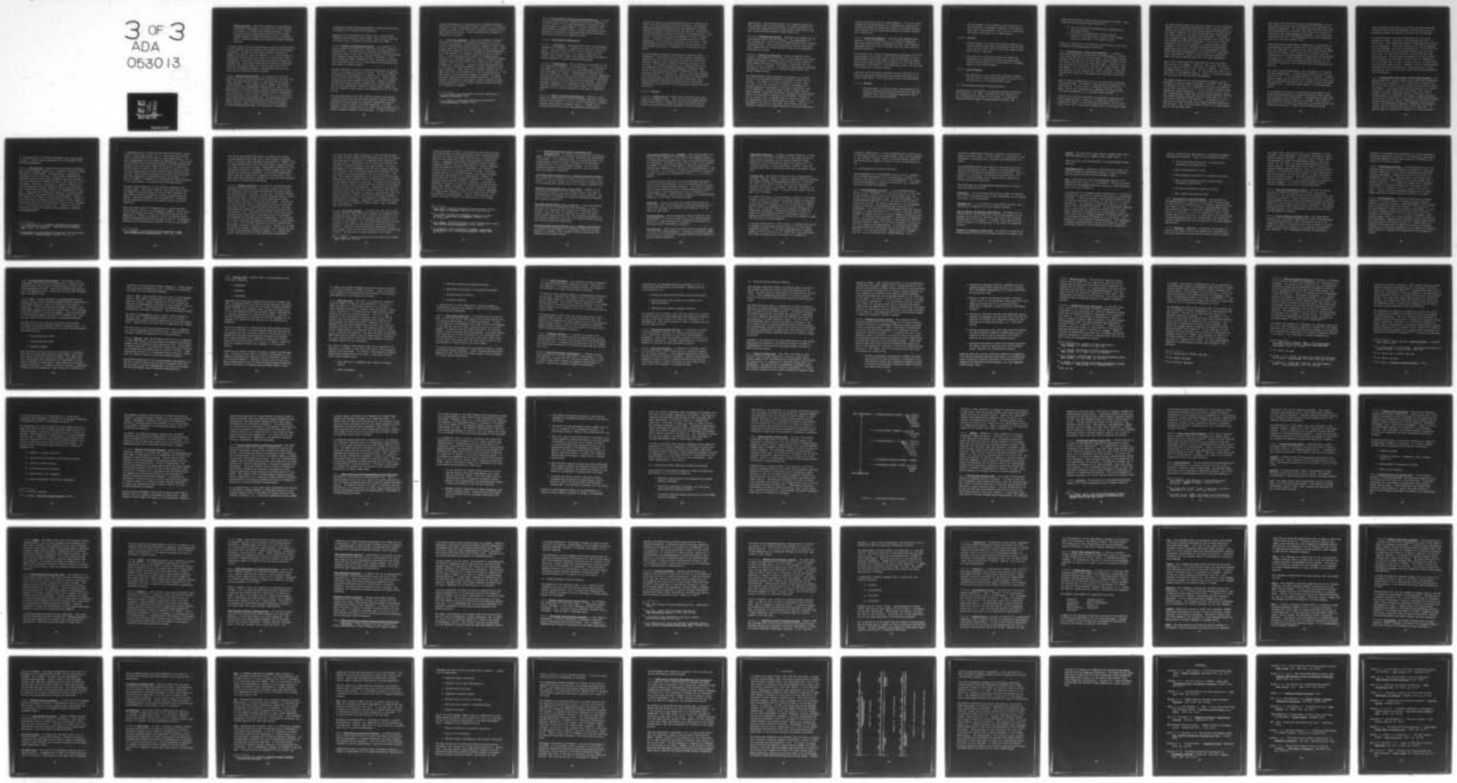
<u>Circuit Switching</u> - This establishes dedicated channels or circuits in support of communications between users and host or between hosts. Channel utilization is limited to traffic between sender and receiver systems. CYBERNET, INFONET, and DATRAN are examples of circuit-switched networks.

<u>Message Switching</u> - Using this technique messages are routed through circuits between sender and receiver to increase circuit utilization. Increased circuit utilization is achieved at the expense of more switching overhead.

AD-A053 013 SPERRY UNIVAC ST PAUL MINN DEFENSE SYSTEMS DIV F/G 9/2
RECONFIGURABLE COMPUTER SYSTEM DESIGN FACILITY INITIAL DESIGN S--ETC(U)
JAN 78 D R ANDERSON, L D ANDERSON, K Y WEN F30602-76-C-0355

UNCLASSIFIED RADC-TR-78-6-VOL-2 NL

3 OF 3
ADA
053013



END
DATE
FILMED
6 -78
DDC

Packet Switching - When this technique is used messages are broken into packets, and packets are routed through the network to shorten the transmission delay. Packet switching allows simultaneous transmission of several message packets through different circuits to circumvent delay. ARPANET and MERIT are examples of packet-switched networks.

In normal network process communications sources and destinations of the communicators are explicitly delineated within messages or packets. The explicit destination numbers (and source numbers) are used to route the message. It is also possible to communicate using process name rather than destination where the physical location of the destination process is known only by deduction. The process technique has functional advantages: processes can migrate among the network components without creating a bookkeeping problem, and redundant process structures can be created for fault tolerance.

5.2.2.2 Dedicated Bus Networks - The dedicated bus network differs from the networks previously discussed in two major ways. First, the dedicated bus system is under the control of the system designer. In the common carrier systems, the communications lines were independently administered. Second, the application is mission-oriented, implying limited function sets and the ability to analytically determine process intercommunication parameters (rates, queues, delays, etc.) as a result of the limited function set properties. The dedicated bus network shares some of the problems and difficulties of the common carrier network, such as the sharing of resources through a limited communications media, the difficulty of exchanging information between processing elements, communications faults and recovery, and process synchronization.

Examples of the dedicated bus networks are: MIL-STD-1553A, Shipboard Data Multiplex System (SDMS), Sperry Univac Data Bus Controller (DBC), and Memory Multiplex Data Link (MMDL).

5.2.2.3 Parallel Interconnected Network - An alternative to busing systems is the parallel interconnect system, typically a matrix switch. The matrix switch provides the capability to switch any input to any output and the capability of simultaneous data transfer through multiple channels. Advances in solid state technology make it possible to use large matrix switches with reasonable reliability at reasonable switching speeds and data rates for digital signals.

The Navy AN/USQ-67 Centrally Controlled Interconnection System (CCIS) is an example of matrix switching. Parallel information is converted at the peripheral or computer to a serial format which is then routed through the matrix switch. Paths through the matrix switch are computer controlled. Dynamic reconfiguration of the switch from an external point (communications) has not been implemented, because it has not been needed, but the CCIS hardware would support the implementation of this capability.

The Data Manipulator Unit (DMU) allows switching to be dynamically controlled from the attached computers instead of being predefined and controlled by a separate independent computer, providing a better dynamic switching capability. The DMU creates a control problem when it is used to allow communications between separate computers simultaneously. The DMU switching, routing, and masking is controlled by the attached machines. Each machine must sense states (masks, activity, etc.) before it commands a change to the DMU state to prevent interference with other machine activities. Static switching will present no problems in this area.

5.2.3 Arrays and Ensembles - Arrays and ensembles (hereafter referred to as arrays) are another architectural form that utilizes interprocessor communications. Because of their SIMD (Single Instruction Stream, Multiple Data Stream) nature, these architectures generally restrict processor intercommunications to direct transfer of information between Processor Elements (PEs). Koczela has presented a PE-PE intercommunications technique in a conceptual design;[4] Illiac-IV implements nearest neighbor communications between PEs (limited to four nearest neighbors as in Koczela).[5] PEs are coupled through registers to other PEs, which are programmed to accept information from their neighbors. Other arrays and ensembles (STARAN, for example) also have some flexibility in PE intercommunication. The SIMD nature of these machines embeds the protocol of interprocessor communications into problem structure. The PE-PE transfers and protocol that regulate those transfers are more closely associated with individual program instructions than with individual processes.

[5] L. J. Koczela, "The Distributed Processor Organization," Advance in Computers (1968).

[6] K. J. Thurber, "Associative and Parallel Processors," Computing Surveys (December 1975).

5.2.4 <u>RCSDF Processor Communications Recommendations</u> - All the processor communications techniques and protocol discussed so far are relevant to the RCSDF development in different ways. Intercomputer techniques appear to be most relevant to the RCSDF emulation hardware design. General assessments of the processor communications techniques and protocol discussed in the previous section are summarized below.

5.2.4.1 <u>Intercomputer Communication</u>

5.2.4.1.1 <u>I/O Channel</u> - Techniques based upon this connection category are fundamental to operations involving multiple computers. Network systems normally use I/O channels and programmed high level protocol. Our research shows that I/O channels will probably remain the dominant method of interconnecting computers in future systems architectures.

5.2.4.1.2 <u>Interprocessor</u> - This technique is available on very few computers and appears to be seldom used. It is limited in the number of processors than can intercommunicate, hence it does not appear too helpful. Interprocessor technique does have the advantage, however, of directness for control operations; the passing of control information or alerting another processor requires no overt action by the receiving processor. The interprocessor technique is not expected to be a dominant method of processor communication in the future because the majority of computers do not include features for this type of communication.

5.2.4.1.3 <u>Memory Connected Intercomputing</u> - Generally, techniques for communications between processes using this structure are well known because of the use of multiprocessing in several military and commercial computer systems. However, distributed systems must modify multiprocessing techniques to account for distributed system resource control, physical distance, and other parameters (access protection, for example). Memory-connected intercomputing has the advantage of sending information directly from processor to processor, avoiding the receiving processor interrupt structure. The disadvantage is that the receiving processor must be programmed to sense a communication. Processor communications through memory will remain fundamental to multiprocessor operations in the future. Widespread practical use of memory-connected intercomputing for distributed architecture is expected in the next five to ten years.

The intercomputer processor communication technique seems fundamental to developing initial RCSDF hardware operational capabilities and is the recommended approach for obtaining a multiple computer system operating capability because of its low risk. I/O channel connections are a conservative method of implementing an intercomputer capability minimizing the effect on existing machine designs (STARAN and QM-1). Interprocessor connections should form a part of some RCSDF hardware unit architecture, or should be of interest as an architecture to be emulated rather than the dominant interconnect form. Memory-connected intercomputing should also be considered, recognizing the difficulties of heterogeneous machines and data structures.

5.2.4.2 Networks

5.2.4.2.1 Common Carrier - Common carrier networks and their usage will increase in the ıuture with the value-added network assuming dominance for commercial systems and usage, and the mission-oriented network being frequently used for military applications. High level protocols will probably become both more efficient and more specialized. Protocol techniques for interprocess communications will become more sophisticated and able to automatically perform complex operations.

5.2.4.2.2 Dedicated Bus Systems - Dedicated bus systems will be increasingly used in military systems. The establishment of Mil-standards will tend to provide standardized protocol at basic interface levels. The distinct possibility exists that higher level common carrier network protocol logic will be adapted to dedicated bus systems.

5.2.4.2.3 Parallel Interconnects - Usage of parallel interconnects will probably be restricted to those applications where higher performance is needed and high costs can be tolerated. Rapid dynamic switching of the parallel interconnect and its control is still a problem. The parallel interconnect technology may appear in the future as an internal design feature of a single computer design.

The RCSDF hardware should probably be considered in the near future as a multicomputer system for emulation rather than network. The number of envisioned machines is small, hence, the need in the near future for a shared interconnect or a switched parallel interconnect as the major interconnect form is not demanding. The architecture forms of common-carrier-based networks, dedicated bus systems, and parallel interconnects need to be understood as objects to be emulated. Protocol at the two lowest levels (interface and I/O structure) need not be incorporated into the RCSDF emulating system. However, it is recommended that some of the higher level network protocol (process-to-process) be studied further as a potential method of process coordination for the RCSDF hardware. It is also recommended that the architecture of a dedicated bus system be examined and studied for medium term usage of the RCSDF facility when expansion may be encountered and point-to-point interconnects restrictive.

5.2.4.3 Arrays and Ensembles - The array and ensemble architecture is still of limited use in the field of computers. When these communications techniques are used, they seem most appropriate for the class of problems that are well suited to SIMD architectures.

The protocols and processor communications techniques for arrays and ensembles can be seen to have limited application to the RCSDF hardware system and remain primarily an emulation object. Two possible areas of application seem to be in the use of the STARAN and in the design of a microprocessor array, but this has to be determined in future studies.

Given the above technology assessment and the relevance of the different processor communications techniques and protocol reviewed, RCSDF needs in this area seem to be as outlined below.

5.2.4.4 Topology

- The development of point-to-point intercommunications between existing and planned devices for the near term including such items as interface adaptation, multiplexing, and interrupt handling.

- The development of the architectural definition of a busing system to accommodate expansion in the medium term, which could provide a first try at a higher perperformance technology (e.g., fiber optics).

5.2.4.5 Protocol

- The development of a process-to-process communication method, perhaps based upon communications network high level protocol logic, which is the total scope of the RCSDF control structure.

- The evolution of emulation methods to reflect the performance of communications protocols and communications network behavior into the emulation of network systems on the RCSDF hardware.

5.2.4.6 Array Technique

- The examination of the techniques (processor communications) used in known array machines for inclusion in the definition of the microprocessor array.

5.3 Microprocessor Network Technical Baseline

The main goal of the RCSDF is to provide an emulation facility and support for all users, including the user intending to put a microprocessor network in his system. To efficiently emulate different types of microprocessor networks, the

RCSDF should provide a generalized microprocessor network. This technical study presents ideas relevant to:

1) how to construct an architecturally flexible microprocessor network.
2) the scope of microprocessor network designs.
3) emulation strategies when a microprocessor network design is submitted to the RCSDF facility.

Below is a discussion of the general state of the art for microprocessors and multiprocessor networking.

5.3.1 Microprocessor - Survey and Assessment - In general, there are essentially two families of microprocessors: the bit-slice family and the one chip CPU family. The bit-slice family includes Intel 3002 (a 2-bit ALU slice), Motorola 10800 (a 4-bit ALU slice), and AMD 2901 (a 4-bit RALU slice). Examples of the one chip CPUs are Intel 8080, Motorola 6800, Fairchild F8, and RCA COSMAC. All of these are 8-bit CPUs. However, 16-bit CPUs are beginning to gain popularity. Texas Instrument's TMS 9900 and National's PACE are good examples of the single chip 16-bit CPUs.

Each of these microprocessors has its own particular advantages and disadvantages , according to what application measures it is subjected to. In this report, no general assessment will be made. Rather, the discussion will emphasize the applicability of these microprocessors in a microprocessor network.

Normally the bit-slice processors are adequately supplied with other supporting parts, such as a microprogram control unit, a microprogram memory, a look-ahead carry generator, and an interrupt control unit. Their main disadvantage is that several chips and other logic devices will have to be used to form a processor. However, bit slices offer flexibility wherein the user can design as wide a CPU as he wants and as unique an architecture as he wishes. Moreover, bit slice processors are currently about ten times as fast as the single chip processors. For normal 8-bit operations where speed is not essential, the single chip CPUs probably provide more advantages. For 16-bit or longer operations, the bit-slice approach is currently more widely accepted. Since one of the most interesting microprocessor applications is to use a network of microprocessors as high-speed main frame replacements, 40 (or even 64) bit operations can be anticipated. Hence, for the flexibility required by RCSDF, bit-slice architectures seem to be more applicable.

Most of the single chip 8-bit CPUs and the 4-bit slice ALUs have 40 pins in a package. Generally, for a 8-bit CPU chip, 8 pins are used for data and 16 are used for address. This gives an address space of 32K to 64K. For bit-slice CPUs, there is no real limit to the address capacity. Hence, for general purposes, the bit-slice CPUs will be more advantageous.

One common feature for all third generation computers is the abundance of general purpose registers in the CPU. This feature is necessary for efficient programming, providing easy addressibility and eliminating a lot of the ALU register congestion and unnecessary memory references. A survey of the currently available microprocessors shows that bit-slice RALUs generally provide more general purpose registers than the single chip CPUs, e.g., 10 for Intel 3002 and 16 for AMD 2901. In comparison, Intel 8080 has 6 and Motorola 6800 has no general purpose register on the CPU chip. Thus, from this point of view, it is more preferable to use bit-slice CPUs.

Most single chip CPUs are not user-microprogrammable, in the sense that the user cannot microprogram his own sequences of hardwired operations. Yet most RALU chip families contain ROM and microinstruction controller chips that allow the user to develop his own microprogrammable processor. For flexibility required by the RCSDF microprogrammable multichip processors seem to be the more logical choice.

There are two main approaches to performing I/O operations in microprocessor systems, as there are in minicomputer systems. The first approach is typified by the PDP-11 and Motorola 6800. There are no special I/O instructions; I/O devices are considered part of the memory address space and are connected onto the memory bus. Another approach is exemplified by the Nova minicomputers and the Intel 8080 which have specific I/O instructions that do not use up address space for addressing the devices.

For single chip microprocessors, the I/O approach is fixed by the chip manufacturer. In multichip processors, the user can choose his own. To use microprocessors in a network, a special network adaptor may be required to interface each processor with the interprocessor connection (bus, crossbar switch, multistage networks, etc.). This may require special I/O handling capability.

To be effectively used in a multiprocessing environment, the microprocessor should use an asynchronous method when communicating outside the microprocessor. A handshaking procedure and some additional control lines will be needed. The processor should also be interruptible and be able to quickly find out the interrupt condition(s). Many bit-slice families include interrupt register and control unit chips, which provide easy masking and bit extraction operations, making the bit-slice processors capable of handling interrupts very efficiently.

As mentioned before, the main disadvantage of a bit-slice processor is that it uses many more LSI chips than the 8- or 16-bit unit. This implies more space is required and more cost for the processor. A bit-slice processor of 16 bits equipped with the sequencer, the carry look-ahead unit, 1K of 40-bit control ROM (or RAM), and the interrupt control unit now costs about $200. A single chip CPU with a control ROM will cost only about $50. So the trade-off between bit-slice processor and the single chip processor is one of cost vs. flexibility. It is projected that in 1980, a 40-pin chip or a 1K byte memory will cost only about $5 each. Therefore, at that time processor hardware cost will become only an insignificant part of the whole system, and the system that gives the better performance will thus prevail.

5.3.2 <u>Interprocessor Connections - Survey and Assessment</u> - All multiple-processor systems require data communication capability between various processors. The interprocessor connection is one of the most crucial factors affecting multiprocessor system performance. The factors governing the selection of a certain connection include communication techniques, data transfer methods, connection bandwidth (amount of data that can be transferred by the connection per second), required bandwidth (total amount of data that is required to be transferred between various processors per second), and expected connection contention (the performance degradation caused by two processors trying to get on the same data path at the same time). In general there are three main categories of interconnections:

1) bus structures, 2) crossbar structures, and 3) multistage networks. Each of these categories will be discussed in the following paragraphs.

5.3.2.1 Bus Structures - Buses have been essential elements in most third generation computers.[6] In general, buses are either dedicated or nondedicated. A dedicated bus is permanently assigned to a pair of devices. The principal advantages of dedicated buses are their high throughput and ease of control. A major disadvantage of dedicated buses is the high cost of all the cables, connectors, and drivers. Moreover, adding a new device frequently involves adding new cables and interfaces. Thus a Dedicated Bus Network (DBN) is not expandable nor flexible. A more practical connection system will use only partial DBN by omitting some dedicated buses between certain devices, thus reducing the number of buses required greatly. However, a lot of time may be consumed in going through a large number of node-devices when the two communicating nodes are not 'adjacent' (i.e., not connected by a direct bus). Examples of partial DBNs are the Illiac IV Network and IMS Associates' Hypercube of Microprocessors.[7]

[7] K. J. Thurber et al, "A Systematic Approach to the Design of Digital Bussing Structures," Proc. Fall Joint Computer Conf. (1972), pp. 719-740.

[8] "Hyperdimensional Microprocessor Collection Seen Functioning as Mainframe," Digital Design (November 1975), p. 20.

A nondedicated bus is shared by many devices (processor, memory module, input-output device, etc.). Each device can transfer data to another device, but only on a time-shared basis. The main disadvantages of this single bus approach are 1) the obvious bus contention for too much data trying to transfer between the devices, and 2) the total system failure due to a single component failure in the bus. When the number of devices is small and the expected amount of data transfer among various devices is small, the single bus structure is the most economical one. Most unit machines are structured around this concept, e.g., PDP-11's UNIBUS. An example of a multiprocessor system is based on the same concept is the Minerva Multi-Microprocessor.[8]

The most logical extension to the single bus approach uses multiple buses. This scheme requires that active devices have the capability of selecting a bus for its data transfers and that passive devices be capable of resolving simultaneous requests. Using this approach, the bus bandwidth is increased, and the redundancy allows the bus system to have a fail-soft capability.

There are three classes of nondedicated bus control methods: daisy chaining, polling, and independent requests. Daisy chaining has the simplest structure, and additional devices can be added easily. However, a failure in one of the devices could prevent succeeding devices from getting on the bus. Furthermore, because of the fixed priority structure, devices at the farther

[9] L. C. Widdoes, "The Minerva Multi-Microprocessor," Proc. Third Annual Symp. on Computer Arch. (1976), pp. 34-39.

end of the chain could conceivably be blocked from the bus. Polling requires more lines, but it can be made to eliminate both of the daisy chain problems mentioned above. However, both of these methods suffer from long time delays, especially when the number of devices is large. Independent requests solve the long delay problem at the expense of more hardware and a more complex bus allocation algorithm. For multiple nondedicated buses , each bus should have a separate bus controller, and in some cases there should be a master bus controller that monitors the status of various bus controllers.

5.3.2.2 Crossbar Structures - To allow for simultaneous data transfers from many devices, a multiport system can be used. The connection allows multiple simultaneous transfers between two exclusive sets of devices, in this case, the memory modules and the processors. Any data transfers required within a set of devices will have to be done in a mail-drop fashion (using common memory locations for delivery and pickup). Each device will have multiple ports to handle devices from the other set. The access control is contained within each device, and dedicated buses are required to connect each pair of devices from the different sets. Since the control logic and the ports are contained within each device, each device has to be designed to accommodate the maximum system configuration. This, however, is in turn limited by the number of ports allowed in each device. As a result, this multiport approach is usually employed where the number of devices in each set is small. Because of the high connection throughput, modified versions of this scheme are used in many third generation computer systems, e.g., UNIVAC 1100 series, IBM S/360 and S/370, HIS 635 and 645, and AN/UYK-7.

To offset the port number limitation of the multiport system, the access control logic and the dedicated buses can be removed from each device and placed in a separate module called the crossbar switch. Each device will now have only one port and the device interface design is thus much cleaner. The main advantages of this arrangement are that the crossbar (and also the multiport) allow simultaneous data path transfers and that the control is relatively easier than for the multibus. However, the biggest disadvantage is its high cost since the gate complexity of the crossbar switch is proportional to the product of the number of devices in each set. To reduce the cost of the switch, partial words can be transmitted serially at a frequency higher than that of the processors. Considering current microprocessor technology, the slow instruction cycle time seems to make this serial transmission feasible. Nevertheless, when the number of devices is large, the switch complexity increases greatly and the crossbar switch represents an expensive way of providing high throughput. Examples of multiprocessing systems that use crossbar switches are Hughes H4400, C.mmp, Burroughs D825, and CDC 6600.

5.3.2.3 <u>Multistage Networks</u> - To offset the high cost of the crossbar network, many permutation networks are suggested since simultaneous data transfers from N inputs to N outputs can be regarded as a permutation of N elements (assuming there is no conflict at the output). The one common characteristic of these networks is that they all have multiple stages of switching elements. Batcher's sorting network has been proven to be a near-optimal solution to the worst case permutation network delay problem.[9] Yet the cost of such a network is still high.

[10] K. E. Batcher, "Sorting Networks and Their Application," <u>Proc. SJCC</u> (1968), pp. 307-314.

Current researchers intend to find networks that would minimize the network delay for most permutations (while penalizing some), hoping to cut down the average network delay. An interesting group of the second class comes from the perfect shuffle connection. Independent researches resulted in Stone's shuffle exchange network[10], Lawrie's Omega network[11], Pease's indirect n-cube microprocessor array[12], and STARAN's flip network[13], which all turn out to be topologically the same. Unlike the crossbar switch or the Batcher network, this shuffle exchange network does not permute all n! permutations. However, it can perform most of the permutations required of a tightly coupled parallel processor system. Moreover, it can also be modified to be used in a Multiple Instruction Stream Multiple Data Stream (MIMD) environment. The main attractions of this network are its ease of control and its low network complexity $0(N \log_2 N)$ gates compared to $0(N^2)$ for crossbar and $0(N \log^2 N)$ for Batcher's sorting network. However, much research still needs to be done to improve the feasibility of this network.

[11] H. S. Stone, "Parallel Processing with the Perfect Shuffle," IEEE Trans. on Computers (February 1971), pp. 153-161.

[12] D. H. Lawrie, "Access and Alignment of Data in an Array Processor," IEEE Trans. on Computers (December 1974), pp. 1145-1155.

[13] M. C. Pease, "The Indirect Binary n-Cube Microprocessor Array," IEEE Computer Society Repository, 75-100 (1975).

[14] K. E. Batcher, "The Flip Network in STARAN," Proc. 1976 International Conf. on Parallel Processing (August 1976), pp. 65-71.

5.3.3 RCSDF Microprocessor Network Requirements and Recommendations - To specify a microprocessor network for RCSDF, we must first outline certain properties and requirements of the RCSDF. It is understood that as an RCSDF equipment, the microprocessor network is used to emulate the multiprocessor network portion of the user-designed system. We will first categorize different multiprocessor networks that the RCSDF should be capable of emulating.

Bus-Oriented Multiprocessor (e.g., Minerva Multimicroprocessor) - The easiest way to emulate this type of system is to use a bus. However, we can also emulate this system using a crossbar switch if we visualize the crossbar switch in a slightly different way.

Multiport Multiprocessor (e.g., Univac 1108) - Because of the topological similarity with the crossbar switch, this type of multiprocessor organization can be fairly easily emulated. The switch will have to perform all selection procedures and memory port conflict resolutions.

Hypercube Multiprocessor (e.g., Illiac IV) - Since the crossbar can allow any set of simultaneous data transfers, all data transfers on a hypercube type network can be emulated with no trouble. When the number of processors is large, the replication method (emulating more than one user-design processor on each facility processor) will have to be used to reduce the high cost and yet retain the versatility.

Multistage Multiprocessor Network (e.g., STARAN's Flip Network) - All permutations performed on a multistage network can be performed on a crossbar, and a crossbar-oriented network can thus emulate any multistage network.

Intercomputer Network (e.g., ARPANET) - Each microprocessor can emulate a host computer at each node of the computer network. But communication procedures will have to be implemented on each microprocessor, and network delay time will have to be compiled and added on by each microprocessor. Moreover, emulating a complex host computer (e.g., IBM 370) on a microprocessor is just not feasible. So although the RCSDF microprocessor network can be made to emulate an intercomputer network, it is not recommended.

It should be noted that each of the user-designed processors can be a wide word processor, can be a pipelined processor, or can even be a special purpose processor. The intent to emulate each of these processors with a single microprocessor places a large burden on the microprocessor, and its flexibility is heavily challenged. Several points regarding the microprocessor selection in the RCSDF are discussed below.

Word Width - The word size of the microprocessor should be sufficiently wide, since emulating a wide word operation on a small word processor will require extra emulation time and emulation code space.

Microprogramming - The microprocessor chosen should be microprogrammable, or should at least have a rich instruction set. Emulating a complex processor on a capability-limited microprocessor may be unnecessarily difficult.

ALU Functions - There should be sufficient ALU functional capabilities in the microprocessor. A variety of arithmetic modes and various bit manipulation functions would be very helpful to general purpose emulation.

Addressable Registers - In general purpose emulation, to keep track of all the timing information in the target processor, many internal registers will be used as emulated clocks. Emulating a wide word operation would also require a lot of internal registers to hold intermediate results. Hence the microprocessor chosen should have sufficient addressable registers.

Control Store - The control stores for most microprocessors are ROMs. However, RCSDF is a general purpose emulation facility and requires dynamic microprogramming capability. Microcodes will have to be loaded into the control store every time a different emulation is in progress. ROMs are, therefore, not suitable for control stores. Writable control memory (such as RAMs) will be needed.

To specify the number of microprocessors required in the RCSDF network, we have to determine the average number of processors in the user-designed systems. Currently existing or proposed systems consist of anywhere from 4 to 256 processors. Therefore, 16 to 32 microprocessors would probably be adequate to handle most emulations. Depending on whether the QM-1 can handle all the bookkeeping of the microprocessor network, a separate microprocessor array controller may be needed to supervise the array of microprocessors.

As we have seen earlier in this section, a crossbar-type switch is capable of efficiently emulating most of the other interconnection networks. So for easy reconfiguration, crossbar appears to be the switch to use. RCSDF has a Data Manipulator Unit (DMU) available which is similar to a crossbar in functional capabilities. If each microprocessor is paired with a local memory module, the microprocessor array can be connected to (and through) the DMU. Hence, by careful design, RCSDF can avoid procuring an expensive network for the microprocessor array.

5.4 Microprogramming Technical Baseline

Microprogramming and microprogrammed devices play an important role in RCSDF development since the fundamental concepts of emulation are based on the use of microprogramming. This report puts microprogramming in perspective and serves as a technical baseline for RCSDF developments.

5.4.1 Microprogramming Overview - A microprogrammed design results in the replacement of a significant portion of the logic of the computer control section by stored program logic contained in a high speed ROM or RAM. The main advantages of the microprogramming approach are the cost savings and the design flexibility allowed. Two general approaches to microprogrammed design are vertical microprogramming and horizontal microprogramming. In a vertically microprogrammed system the instruction word is relatively narrow (i.e., 16 bits), and each field is encoded in a manner similar to a normal computer instruction word format. The discrete control present in horizontal microprogramming is relinquished for compactness in representation; hence it typically requires several instructions to perform a given function with vertical microprogramming. In horizontally microprogrammed machines the instruction word is extremely wide (up to 360 bits). In this architecture, functions are bit encoded; each bit in the instruction has a unique meaning. It is possible to enable many functions in parallel with one horizontal microinstruction. In cases where it is possible to take advantage of the parallelism, extremely good performance can result.

Microprogrammed machines can operate at extremely fast microinstruction execution times. Speeds of 100 nanoseconds are typical with ECL machines operating at 50 nanoseconds. This is possible not only due to the overlap incorporated into the design, but also because of high speed control memories and limited instruction complexity.

Some advantages of microprogrammed implementation of control logic are described briefly.

Flexibility - Critical design decisions can either be deferred until late in the design phase or once implemented can be changed by writing a new microprogram.

Changeability - It is possible to reconfigure a machine, once installed, by changing the control memory contents.

Ease of Design, Maintenance, and Checkout - The design of the control section can be shared with the microprogrammer who implements the instruction repertoire and much of the interrupt handling work of the machine. A cleaner, more logical design usually results in a machine that is also easier to debug and maintain.

Extension of Machine's Useful Life - The ability to enhance and change machine functions via microcode extends its useful life.

Economy - The bulk of the logic used is standard ROMs and/or RAMs and therefore less expensive than random logic.

There are also a few disadvantages of microprogrammed design. They are:

Performance Loss - A decrease in performance can result if a general purpose microprogrammed machine is used to emulate a specific machine without special hardware support.

Cost - Selecting the wrong microprogrammed machine for an application will result in a cost penalty. Using an extremely powerful microprogrammed machine to emulate a simple architecture is not cost effective.

A further use of a microprogrammed architecture is to allow the tailoring of a machine to a specific application via customized microcode: a user may procure a basic machine capability, and by modifying and adding microcode, he may adapt it to a special application. This approach is especially attractive if the machine has a writable micromemory. If the application requires such a large amount of specialized firmware that the micromemory capacity is exceeded and if the performance requirements allow time for swapping segments, microcode may be read in from back-up storage as demand for its use occurs. This capability is called dynamic microprogramming. The advantages of dynamic microprogramming include cost tailoring, storage requirement reduction, and increased functional capability.

However, there are also many pitfalls in providing a dynamic programming environment. Issues to be considered include:

- o Conflicts between flexibility of customer microprogramming and compatibility
- o What microprogramming is best
- o What debugging aids should be extended to the user
- o What are multiprogramming and multiprocessing environment ramifications
- o How should engineering changes be handled
- o What loading procedures are best

5.4.2 Applications of Microprogramming - The potential of microprogramming as a means for altering system architectures to allow efficient implementation of a specific application has been known for years. In its infancy microprogrammed design was considered primarily a means for implementing the control section of a computer. It is now considered to have a vastly broader application than this. In this section the applications of microprogrammed design will be discussed. The traditional emulation application will be reviewed as well as more advanced applications.

5.4.2.1 Emulation - Emulation is defined as the process of executing the object code of one computer on a different computer and obtaining the same logical results without changing the object code. This often results in a degradation in performance. Microprogramming is the common technique for implementing emulation machines. Read-only control memories can be reconfigured by inserting preprogrammed modules. Read-write control can, of course, be easily changed dynamically under machine control. Two approaches can be taken in designing a microprogrammed emulating machine. One is the general purpose machine. Its architecture allows relatively easy emulation of many architectures, but usually results in poorer performance than the emulated machine. The other approach is to design a special purpose emulator. It is initially aimed at emulating a specific computer. It includes special purpose hardware (emulation adapter) and results in high performance.

5.4.2.2 Data Base Management Processor - One way of handling the task of database management is to place a processing capability at the interface between the mass storage device and the central processor. Being a specialized processor, it would accept commands from the central processor and perform relatively powerful or time consuming functions on the database that would relieve processing load from the central processor. The specialized DBM processor can be implemented relatively easily with a microprogrammable processor.

5.4.2.3 Virtual Address Translator (VAT) - The Experimental Processor for Intermodular Communication (EPIC) microprogrammed Virtual Address Translator (VAT) developed at Univac is an independent address translating device which provides a virtual address environment by mapping virtual address into extended real addresses. VAT microsequences perform the operations necessary for implementing and managing the virtual addressing capability supplied by the device. The flexibility provided by a microprogrammed VAT allows VAT operations of differing complexity to be easily designed.

5.4.2.4 Performance Monitoring - The microprogram level is an excellent point within the computer's structure for capturing valuable information for use in both debugging and system performance measurement. Measurements taken at this level are transparent to the user. They do require some machine time, but do not require a modification to the user program such as software hooks do. It is relatively inexpensive to implement measurements at this point because it is basically a microroutine that is inserted in line at the point monitoring is to occur. Finally, when the measurement programs are no longer needed they can be removed from the system with no impact.

5.4.2.5 Microdiagnostics - There has always been an inherent difficulty in using software to diagnose hardware failures because it typically requires a large percentage of the hardware base to execute even basic instructions of the repertoire. If the microdiagnostic program was carefully constructed, it could be used to diagnose certain failures using only a small percentage of the hardware. It is, of course, necessary to have enough hardware functions to allow fetching of microinstructions from micromemory, keeping track of current program counter value, and operating from the basic register set. Beyond that, the hardware can be exercised to search for failures by carefully building up the functional capability of the machine as function after function is checked and verified by the microdiagnostic program.

5.4.3 Microprogrammed Architectures - Some typical microprogrammed architectures are examined here to determine their characteristics and to gain an understanding of some of the design tradeoffs involved. Three basic machine architectures will be discussed briefly. These are the Univac MPC, Nanodata QM-1, and IBM 360 line.

5.4.3.1 MPC - Univac's approach to microprogrammed machine design utilizes the MPC (Microprogrammed Computer) architecture. The primary function of MPC is to emululate the AN/UYK-20 repertoire. MPC was specifically designed to provide a high performance emulation of the UYK-20 repertoire. Some functions within the emulator are done by hardware. Functions performed by the emulation adaptor include format decomposition, status setting, control and interpretation, and operand fetching.

The MPC-1 has a 16-bit wide microinstruction word format, utilizing a vertically encoded, register oriented format. It has three primary hardware features to provide high performance. These features are:

- Microinstruction overlap
- Microinstruction repeat
- Emulation adaptor

MPC-2 and MPC-3 are quite similar to each other in structure. MPC-2 has a 36-bit microinstruction word length, while MPC-3 has a 40-bit microinstruction word length. The impetus for their development was a recognition for enhancement in five areas of the MPC-1 architecture. These areas were increased register addressing, increased accumulator selection, additional conditional branch capability, more convenient introduction of constants, and more parallel control capability. These improvements were all accomplished in MPC-2 and MPC-3 by introducing the wider word and associated hardware.

5.4.3.2 QM-1 - QM-1 operates under two levels of microprogram control; this offers the advantages of both horizontal and vertical control. Machine instructions in main storage are executed by microprograms contained in control memory. This is vertical control. Microinstructions are executed by nanoinstructions contained in nanostore. This is horizontal control.

The level of programmable control closest to the hardware is the nano level. Nanoprogramming defines a set of control sequences that supports the microinstruction of the next higher level. Typically the nano level code is supplied by the vendor.

The next level is the microprogramming level and is supported by a fully readable/writable control store. Microroutines are written to match the requirements of each application.

5.4.3.3 IBM 360 - When IBM introduced the 360 line of computers in 1964, they hoped to provide upward and downward architectural compatibility to commercial data processing users. They wished to provide efficient, cost effective, timely tools for conversion of user application programs from one system to another. These requirements resulted in a fundamental design decision to implement the control section in microcode whenever possible.

The primary beneficiary of the microprogrammed implementation of IBM 360's control section was the small and medium scale user who was given a rich and comprehensive instruction set without the high development cost of hardwired implementation.

5.4.4 Software Aids - Software aids to microprogramming fall into three categories:

- Assemblers
- Compilers
- Simulators

Assemblers are fairly well known to most users of computer systems. An approach taken at Univac for many machines, including microprogrammed ones , is the use of a General Purpose Assembler (GPA). The GPA is an assembler that utilizes the capabilities of the 1108 assembler and allows specific computer descriptions to influence generation of instructions and data. With GPA, a user has essentially all the power of the 1108 assembler at his disposal at a fraction of the typical cost for developing an assembler.

The use of compilers for converting high level program descriptions into microcode is uncommon because compilers typically do not generate the most efficient version of the machine code, but degrade the high speed implementation advantage of microcode. For a system targeted for an application requiring a sizable amount of custom microcoding at the site, a compiler capability with its accompanying high level language is an attractive feature.

Simulators are another software aid used to support microprogramming. Simulators can be used to interpret microprograms of one computer and perform the functions (at the instruction level) of the simulated computer. Simulation allows a user to test and debug microprograms before the object hardware becomes available. Another advantage of a simulator during program debug is that it offers more powerful debugging aids than would be available on the object machine, at least initially. A major difficulty with simulation is verifying that the simulator is an accurate representation of the hardware of interest.

5.4.5 RCSDF Emulation - The RCSDF is intended as a research facility capable of general system emulation. To do this, it needs the ability to efficiently emulate a wide variety of target machine organizations, whether they are obsolete, existing, or innovative. To accomplish this, there must be a soft architecture for the emulating machine. This is in sharp contrast to the hard emulating machines (such as the IBM 360 models) which are designed to interpret only a small set of architectures with great efficiency. Machines such as the Nanodata QM-1 and the Burroughs B1700 are built with such soft emulation architecture in mind. The Stanford Emulation Laboratory's EMMY System is an emulation facility which is intended to be a universal host machine. The main difference between the EMMY System and the RCSDF is that the EMMY System is concerned with only conventionally structural target machines; multiple processor systems are not considered.

The range of computer architectures that can be expected to be emulated on the RCSDF is much wider than those of the emulating machines mentioned above. As a result, more supporting hardware features are needed than for the others. There are six main RCSDF related capabilities:

- Easy operation on embedded state images of emulated machine

- Memory management

- Efficient instruction decoding structure
- Simplified configuration of emulated environment
- Multiple-machine emulation
- Data path emulation

To achieve each of these capabilities, certain hardware or architectural features are required. They will be discussed in the following paragraphs.

5.4.5.1 State Image Operations - An emulation facility has to map the data and control state images of the target machine into the existing state images of the emulating machine. It then has to operate on the embedded state images of the target machine in the same way that the target instruction does on its state images. To perform these two tasks efficiently, the facility should be able to provide easy image mapping. One useful feature that would further efficient emulation is the multidata word size addressability (e.g., byte, half word, word, double word). In some cases, high speed shift and mask capabilities will be very helpful to access unusual size state images.

It would be most advantageous for the emulating machine to have many general purpose registers. Target working registers can be mapped onto them, and slow main memory access is not necessary for target register operations.

5.4.5.2 Memory Management - The problems of memory management in a generalized emulation system are twofold. The first is the handling of the emulation program storage; the second is the handling of a target machine memory hierarchy.

To solve the first problem, dynamic microprogramming techniques can be used. Emulation code segments will be rolled in from the secondary memory to the writable control store by demand paging techniques. This allows different emulators to be loaded and replaced in the emulating machine very quickly. It also allows larger control programs to be run on a relatively smaller control memory.

The second problem occurs when the target machine has a hierarchical memory system. The facility should then have the capability of mapping each target memory level into the emulation facility memory system. Hence, multilevel memory system and complex address mapping functions are needed in the facility to handle the memory management emulation.

5.4.5.3 Efficient Decoding - To increase the speed of the emulation process, the emulating machine should be able to have microihstruction look-ahead, fetching possible microinstructions for the next step. A separate microinstruction unit will enable the simultaneous operation of the function units and the instruction unit.

5.4.5.4 Target Environment Configuration - A machine environment consists of 1) the data and control state images, 2) a set of primitives to modify and test the state images, and 3) a set of control rules which decides the sequence of primitives to execute. The facility should be capable of statically reconfiguring the executing machine environment so that it matches the emulated machine environment. Examples of such reconfigurations are:

- o Setting up gating paths between registers and buses.
- o Setting up data word lengths for arithmetic and memory operations.
- o Specifying the number of general registers.

The residual control method, which utilizes control bit vectors to specify each of the above-mentioned reconfiguration patterns, will be a reasonable solution for efficient environment reconfiguration. This control method is used in machines such as the QM-1 and the B1700.

5.4.5.5 <u>Multiple-Processor Emulation</u> - To efficiently emulate a multiple-processor system, the facility should provide at least a subsystem consisting of some fully interconnected processors. Larger multiple-processor systems will have to be emulated by replication. Multiple memory modules will also be needed for efficient multiple-processor emulation.

5.4.5.6 <u>Data Path Emulation</u> - To emulate all the data paths in the emulated system, RCSDF should have full connections among all its system components. Only selected connections would be activated during any specific emulation. The data path emulation capability should be able to represent explicit and implicit architecture paths in order to provide the full range of emulation levels anticipated to be needed by users.

5.5 Operating System Technical Baseline

This document describes research and design efforts related to the operating system and control functions required for the RCSDF. The host facility requirements were described in sections 2 and 3. This paragraph concentrates on a concise identification of the specific RCSDF facility operating system requirements.

The RCSDF operating system is defined as the logic provided in hardware or software necessary to maintain control of and provide user interface with the RCSDF resources. Two different operating system architectural structures are described in paragraph 5.5.1 with suggested modifications and features that should be considered for the RCSDF operating system. The two differ in the methodology used in controlling facility resources. The first assumes that the operating system controls the resources. The second assumes resource control is provided by user processes or processes supplied by an RCSDF staff, the interface of which is made possible through standards enforced by a baseline operating system.

Included in the discussion are brief descriptions of some of the requirements necessary for the RCSDF environment followed by recommendations that should be considered before choosing an RCSDF operating system.

5.5.1 <u>Trade-Off Evaluations</u> - The number and scope of capabilities that have been identified in previous RCSDF design studies suggests that the RCSDF control requirements could assume the proportions of a comprehensive operating system when fully implemented. The descriptions in subsequent paragraphs define two different approaches that should be considered. It is felt that either technique could be used to develop an adequate RCSDF operating system. They differ essentially in the implementation methodology and overall capabilities. Considerations for suggesting the processing element control method are based primarily on the economics of being able to utilize existing resource control capabilities provided by the support software packages normally delivered with the processing elements to be installed at the RCSDF. The facility control method is typical of current R&D control developments. It is recognized as having a greater risk factor, but the gains which can be realized from system flexibility and adaptability to various configurations are anticipated to be significant and are expected to pay larger dividends in the long run. The descriptions are brief but detailed enough to convey the different concepts involved.

5.5.1.1 <u>Processing Element Control</u> - This method of RCSDF operating system control can be thought of as resource control functions performed by independent operating systems contained in each processing element. Each is dedicated to maintaining control of the resources assigned to or configured with the individual processing element. It is assumed that most of the hardware elements installed at the facility would be provided with operating systems containing resource control capabilities designed specifically for the processing element. These operating systems would then need to be modified and enhanced to provide the adaptability necessary to allow for system test and emulation in the RCSDF. Some of the modifications that would be required are listed below.

- Provide interelement communication software for those cases where more than one element is required in the configuration and/or when additional elements are used to provide the environment simulation needed to drive the system being tested.

- Incorporate processes correctly interfaced with the operating system being modified that will interface with nonstandard peripherals in the manner specified by the facility user.

- Enhance or add to the operating system's resource control logic for the performance data extraction tools. These tools would have to be accumulating data sufficiently accurate to enable realistic performance analysis.

- Provide for low-level interface with operating system handlers of peripheral devices that would allow for the execution of systems that themselves assume the characteristics of operating systems, i.e., real time control.

- Provide translation logic that would transform operating system function requests (standardized and produced by an HOL or EDL on the host) into formats required by the associated operating system.

- Establish and maintain control and release discipline for shared resources when more than one processing element requires the configuration resource.

Complete operating systems containing capabilities compatible with existing operating systems would need to be developed for those processing elements installed at the facility but not supplied with operating system software, e.g., the suggested microprocessor array.

5.5.1.2 Facility Control - This method of operating system control can be thought of as a supervising executive that is responsible for monitoring the control of all resources configured with the system. It is constructed of resource control functions that are basic to all operating systems. This concept calls for the concurrent execution of many "units," each interfacing with specific system resources. Some of the units themselves may be unmodified versions of complete operating systems.

5.5.1.2.1 Decomposition Unit Description - In the TSDC evaluation (Section 2), the concept of system decomposition was introduced as the methodology that should be employed by RCSDF users when they define systems or subsystems to be emulated on the facility. This methodology of decomposition, used as a means of reducing complex computer systems to an understandable and describable level, has been recognized and addressed by Horning,[14] Hansen,[15] Dijkstra[16] and Parnas.[17] Further work in software system development is being pursued by the Ballistic Advanced Technology Center,[18] Huntsville, Alabama. BMDATC is involved in establishing guidelines for a decomposition methodology by attempting to formalize a language (RSL) that will aid users during the decomposition process.

[15] J.J. Horning and B. Randell, "Process Structuring," Comp. Surveys V:1 (March 1973), pp. 5-30.

[16] P. B. Hansen, "The Nucleus of a Multiprogramming System," Comm. of ACM XIII:4 (April 1970), pp. 238-242.

[17] E. W. Dijkstra, "The Structure of the Multiprogramming System," Comm. of ACM XI:5 (May 1968), pp. 341-346.

[18] D. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules " Comm. of ACM XV:12 (December 1972).

[19] BMD, op. cit.

In general, the process of decomposition must be performed by system designers aided by requirements specifications, software process design (HOLs) and/or emulation design languages that are available to the designer on a host facility. A more complete discussion of the capabilities of the languages, their uses, and generated outputs is the topic of paragraph 5.7. The result of the decomposition process will produce a system specified by a set of interrelated decomposition units.

The decomposition unit is the smallest part of the system that can be designed to execute independently of other parts of the same system. The unit can be defined as the nature of a processor executing a program, i.e., 1) the program, 2) the machine states which that program might assume during execution, and 3) the data that is associated with the program. This definition is essentially the same as that attached to the motion of a process given by Horning and Randell[19] and used also by Hansen[20] and Dijkstra.[21] For the remainder of this document the term "process" will be used as an equivalent for decompositional unit, element, and processes (in Section 3). The term "program" is used to define the action taken by a processor on a specific set of data. It can be performed by sequences of instructions or actual logic embedded in the processor.

[20] J.J. Horning and B. Randell, op. cit.

[21] P.B. Hansen, op. cit.

[22] E.W. Dijkstra, op. cit.

5.5.1.2.2 RCSDF Environment Description - Systems or subsystems to be tested on the RCSDF are represented by a configuration of processes. The system is thus viewed as a set of interacting, concurrently executing and cooperating processes designed to serve the needs of each other and of the total system. The static combination of the set of processes can be accomplished in the host facility. The dynamic combination, i.e., making resources available to executing processes, is the responsibility of the facility control baseline operating system. The baseline operating system itself must permit this construction of processes (satisfying the user system requirements) by providing the basic operational control functions (algorithms) required. The implementation of this type of operating system is referred to as a kernel or system nucleus. Techniques for the design and implementation of such systems have been described by Bayer,[22] Hansen,[23] Wulf,[24] and Wilhelm.[25]

The memory requirements for individual user processes and/or the set of all user processes is expected to exceed the total memory available from the individual processing elements of the facility. This necessitates implementation of a memory management function in the kernel. Many techniques of memory management are

[23] D.L. Bayer and H. Lycklama, "MERT - A Multi-Environment Real-Time Operating System," ACM Proc. of Fifth Symp., O.S. Princ. (1975), pp. 33-42.

[24] P.B. Hansen, op. cit.

[25] W. Wulf et al, "HYDRA: The Kernel of a Multiprocessor Operating System," Comm. of ACM XVII (June 1974), pp. 337-345.

[26] N. Wilhelm, D. Pessel and C. Merriam, "The CERF Computer System," Proc. of the NCC (1976), pp. 765-768.

described in current literature. Most resolve the memory availability problem by implementation of the concepts of virtual memory as discussed by Denning.[26] A complete description of the functional requirements of a memory manager for an experimental kernel operating system is provided by Kinney and Christiansen.[27] However, the impact and unique characteristics of associative memories (if included as a part of a system being tested) must be understood before virtual memory management for the RCSDF facility control operating system can be defined.

The algorithms or capabilities required to control the use of the processor configured in the RCSDF include process scheduling and dispatching. These are invoked by user processes or operating system processes when 1) stimulated by external sources such as peripheral interrupts, 2) stimulated by demands for memory, and 3) responding to interprocess communication requests. Unique characteristics of process control imposed by the need to synchronize process execution in a multiprocess environment are discussed by Horning[28] and Hansen.[29,30] A promising methodology

[27] P.J. Denning, "Virtual Memory," Computing Surveys (September 1970), pp. 153-189.

[28] L.L. Kinney and B.P. Christiansen, "Functional Requirements of a Memory Manager," Univac PX 1174 (May 1976).

[29] J.J. Horning and B. Randell, op. cit.

[30] P.B. Hansen, op. cit.

[31] P.B. Hansen, Operating System Principles (1973).

for the synchronization of simultaneously or concurrently executing logic sequences is provided by the semaphore mechanism introduced by Dijkstra[31] and expanded by Hansen.[32]

The philosophy of a kernel operating system suggests that the algorithms that provide the above described capabilities do not impose strategies or policies concerning the use of system resources. This becomes the responsibility of the user processes. From the user's standpoint, the algorithms can essentially be viewed as instructions that allow user processes access to the facility resources, such as memory space and processors. Examples of such instructions (called primitives) include the capability to:

1) request or release resources

2) send or receive messages to/from other processes

3) create or delete processes

4) activate/deactivate processes

5) create and/or test semaphores

6) acquire peripheral device state information.

[32] E. W. Dijkstra, *op. cit*.

[33] P. B. Hansen, *Operating System Principles*, (1973).

The processing elements installed for the RCSDF might also be required to emulate different elements of a distributed processor system. Peripherals involved in this type of configuration should be considered global and available to any or all executing processes, regardless of the processing element that is performing the execution.

In summary, the RCSDF environment requires that the RCSDF operating system assume the responsibility for providing basic control mechanisms for all the resources in the facility. The mechanisms that provide these capabilities are collected into a hardware/software set of processes identified as a kernel.

5.5.1.2.3 Facility Control R&D Programs - Sperry Univac is currently involved in a several year research and development program aimed at identifying a set of mechanisms that should be embedded in a kernel operating system. Parallel efforts have resulted in the establishment of disciplined software and configuration control procedures. Using this methodology, the project has developed a prototype kernel operating system capable of providing resource control for one or more processors interfacing with a variable set of peripherals and has demonstrated the capability by maintaining control of two concurrently executing operating system-type processes. The project has progressed to the point where it is felt that the kernel operating system concept is a viable and highly feasible approach to be taken when implementing global resource control in a distributed processor environment.

Similar kernel development efforts are currently being investigated in both the academic and industrial communities. Typical examples include HYDRA, the kernel of a multiprocessor operating system being developed at Carnegie-Mellon University; MERT, a multi-environment real-time operating system developed at Bell Laboratories; and Distributed Computer Network (DCN) being developed at the University of Maryland. The identification of a universal set of functions to be provided in the kernel is currently recognized as a problem since the algorithms are closely related to the characteristics of the resources being controlled and the philosophy used. It is felt that for the RCSDF, a complete set, adaptable to the different resources and user requirements, should be developed.

5.5.2 Special RCSDF Operating System Requirements - The RCSDF facility will consist of a number of processing elements supported by peripheral devices, some standard and some unique. Further, the processing elements are expected to have different basic architectures, with at least one element capable of emulating multiple architecture definitions. The term "reconfigurable" suggests that the RCSDF should provide rules governing the interconnection of various configurations of processing elements and peripherals. This would permit users to build a configuration that closely resembles the construct anticipated for their specific deployable system during testing and emulation on the RCSDF. This requires that the RCSDF operating system provide control capabilities for a potentially large number of different configurations of processing elements and which are capable of emulating a variety of processor architectures.

It is expected that in some cases fully constructed test versions of deployable software systems or subsystems destined for performance analysis runs on the RCSDF will assume control of the facility resources, e.g., a real-time command and control system.

In other cases, the system to be designed will assume these capabilities normally provided by an operating system. In either case the RCSDF operating system will be required to perform the direct interface with the real RCSDF peripheral device components and provide memory management capabilities. For those configurations that require deployable processor elements to be added to the facility, provisions must be made to allow for data and control logic interface.

An extensive emulation/simulation capability will be required if a realistic representation of the deployable system environment is to be made available during RCSDF system emulation and test. The RCSDF operating system must be capable of collecting simulation data as actual data while various transfer rates are demanded and produced by the emulated system. This implies a need for mass storage data management, capable of providing high speed access and of supporting large data bases. The topic of data base management has been examined and the results are being published in another document. The extent of the operating systems involvement will be limited to the interface required with the mass storage media.

5.5.3 <u>RCSDF Operating System Recommendations and Conclusions</u> - The RCSDF is to be used as a system analysis tool. The intended purpose of the tool is to permit the testing and extraction of system performance data while emulating target deployable systems. The expected diversity of both the resources and user applications to be emulated on the RCSDF imposes unique facility control requirements that must be incorporated in the design of an RCSDF operating system.

The current concept of the RCSDF suggests that deployable systems will be developed on a host facility and subsequently tested on the RCSDF. Interface logic, i.e., a description of the resources as assigned to the deployable system, is emulated in the RCSDF. Only in the RCSDF will the required hardware (system resource) interconnection occur. The resource interconnection, while defined during development on the host, must not require the users to be concerned with the controlling philosophy of the resources.

It is important, then, to be able to minimize the number and complexities of the interfaces and also clearly specify their characteristics. Implementation of this can be done by establishing a higher level interface between users and the RCSDF operating system. The methodology currently being adopted to address these requirements will allow the flexibility desired for RCSDF and is inherent in the kernel operating system. It is therefore recommended that the project consider the development of a total facility control type of operating system for RCSDF. Some of the primary reasons for recommending this operating system architecture are listed below.

- It will provide for clearly defined points of intersection between the system development tools (host) and the test tool (RCSDF). This will allow users to establish policies of resource utilization without having to be concerned with the characteristics and/or control requirements of the facility resources.

- Resource control algorithms for memory management and virtual storage control, contained within the kernel, can be optimized for the resources provided at the facility.

- The impacts of adding new devices to the facility complex for performance analysis would be isolated in the kernel.

- The kernel can provide global control capabilities of all or a subset of the configured resources, i.e., processing elements, memory, mass storage devices, etc.

- Since the control algorithms would be localized in the kernel instead of being distributed in the test system or embedded in various pieces of support software, critical performance data extraction points can be established either in the hardware or software. In addition, the performance data related to resource control would be simpler to isolate; once traps are defined they would remain constant for all systems being tested.

- Once the kernel operating system has been established, other support software and operating systems designed for and delivered with the processing elements at the RCSDF can be adapted to interface with the kernel.

- As RCSDF utilization evolves, additional support processes such as facility initialization, control, and file control can be provided by developing a second level, user available interface with the kernel.

Pursuit of investigations leading to the development of a facility control operating system for RCSDF is recommended.

There are several assumptions made throughout the document as to the type of environment expected for the RCSDF. From these assumptions and the associated studies we have concluded that a basic requirement is to provide control or, perhaps more explicitly, to provide the means for users to control use of the resources available in the RCSDF. Information published by others concerned with operating system design philosophy indicates that similar studies have been accomplished and implementation efforts currently are under way. The techniques involved are not new; however, the adaptation of these techniques to an RCSDF environment might in some cases be unique.

In order to provide a more explicit operating system capability definition or to better understand the effects of unique characteristics which might be imposed by the RCSDF, additional information must be garnered. This required information can best be summarized as 1) the characteristics or architecture of the RCSDF hardware and 2) the nature of user requirements. (The latter necessitates a user profile.)

5.6 Distributed Systems Organization Technical Baseline

The presence of distributed systems as a usable and practical architecture is of interest to RCSDF as:

- o Potential architectures to be emulated on the RCSDF emulation hardware.
- o Potentially applicable techniques for controlling RCSDF emulation hardware elements.
- o A potential defining system organization for the RCSDF emulation hardware.

Unfortunately, the popularity of the phrase "distributed processing" has become so great that now almost any computing complex containing more than one processing element capable of simultaneous operation is being called a distributed processing system. The spectrum of distributed processing architectures can be divided into five categories, as shown in Figure 5-1. Distinguishing characteristics are identified for each of the categories and some of the more prominent or advanced mechanizations are identified and discussed in subsequent paragraphs.

5.6.1 Remote Network Systems (RNS) - The term "distributed processing" first found its way into accepted usage in the industry as a description of the affiliations of geographically separated, large, central computing facilities of which ARPANET, TYMNET, MERIT, CYBERNET, and the Lawrence Livermore OCTOPUS are typical examples. The advantages afforded include access to resources which might not otherwise be available such as various language translators, data files built by others, use of procedures, access to storage space, access to specialized processing hardware, and processing time.

Communications in these networks is an extension of the hierarchical message-switching trees which serve timeshare terminals. Communication devices include the lines, customarily common carrier, as well as the communication processors provided by the network and the circuit switching elements provided by the common carrier. Communication processors are used to concentrate messages from terminals, to message switch, and to provide front-end processing for the hosts. Communication protocols accommodate the several terminal protocols, host to host protocols, and network protocols. Store-forward and packet switching communication strategies are employed.

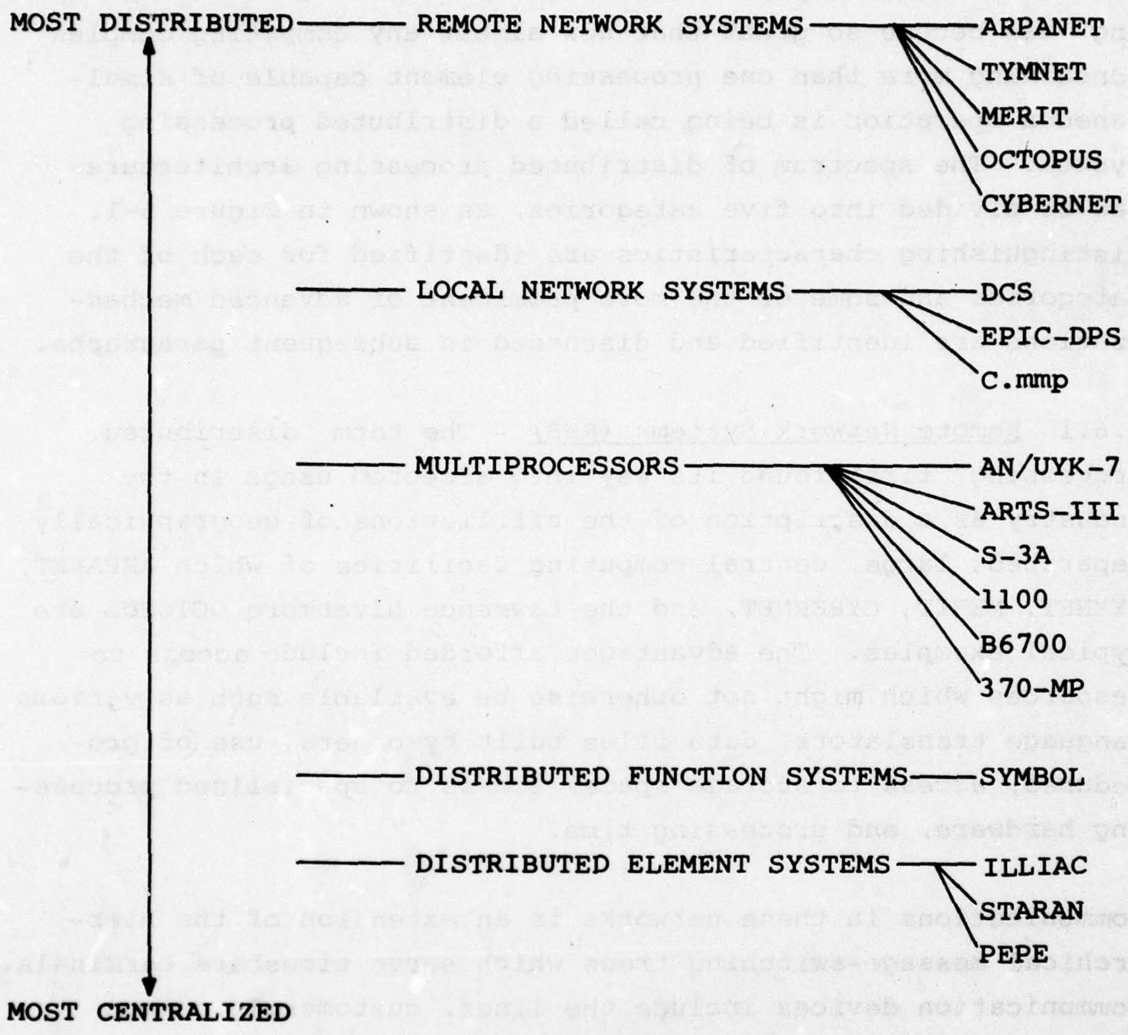


Figure 5-1. Distributed Systems Taxonomy

Control in these coalitions of systems is exercised autonomously by each host. Each host treats traffic from other hosts approximately the same as terminal traffic. There is no interaction between tasks executing on separate hosts, and there is no sharing of resources between hosts. Incoming requests from a terminal for service on a remote host are simply passed along to the communication network. Tasks in execution on any given host simply do not interact with tasks on another host.

5.6.1.1 ARPANET - ARPANET is a loose federation of totally independent computing centers. The reason for its existence is the sharing of procedures, data, and computing machinery among the users who might be located anywhere in the network. Each host is connected to an Interface Message Processor (IMP) Honeywell 516, and these IMPs are themselves connected to each other through a nation-wide interconnection of private telephone circuits. Each IMP has a specially designed interface which adapts it to the circuitry and protocols of the local host. All traffic throughout the network is composed of 1000-bit packets, which are composed, routed, received, and decoded by the IMPs. Totally heterogeneous and autonomous modules are, nevertheless, connected and communicate compatibly through the services provided by a network of standardization modules, the IMPs, which employ hardware and software devices to provide adaptation.

5.6.2 Local Network Systems (LNS) - This class of distributed systems is somewhat similar to the RNS. The class of LNS also has a separately identifiable operating system for each processor, even though the whole collection of operating systems may be regarded as a single operating system. The key characteristic of the RNS and LNS that distinguishes them from the rest of the distributed systems is the term "network." In each RNS or LNS there exists a network through which the processors (or computers) communicate with each other, utilizing a two-party, cooperative, communications-type protocol. Intelligent terminal systems are excluded from the RNS and LNS classes because different nodes do not have the same demanding power (in terms of master/slave relation). The key difference between the RNS and LNS is the distances between nodes. RNS applies to geographically remote systems, while LNS applies to more local connections. Another major difference is that each node in the RNS is a self-contained processing center including computer, memory, and a suite of peripherals, while LNS units may be shared by the network.

5.6.2.1 Distributed Computer System (DCS) - The DCS consists of a number of minicomputers connected by proprietary circuits using T1 carrier technology.[33] The communication circuitry is connected in the shape of a ring, and fixed length messages flow in one direction around the ring until they arrive at the originator, who is responsible for their removal. The actual connections to the ring are communications oriented devices called ring interfaces. Each processor provides its own adaptation to the standardized ring interface. DCS addresses messages not to locations or computer but to other programs. Each ring interface contains a list of programs that are resident at that site, and when a message to one of these programs is recognized, the message is interpreted and executed. This allows programs to migrate throughout the network without affecting software format.

5.6.2.2 EPIC-DPS - The EPIC-DPS is a distributed processing system that is being developed as part of Sperry Univac's

[34] D.J. Farber, et al, "The Distributed Computer System," Proceedings IEEE Computer Society International Conference (March 1973), pp. 31-34.

distributed systems research efforts.[34] The important design features and objectives of this system include incremental expandability of the network with no degradation to the existing resources, resource sharing throughout the network, computational processing at the local level, system-wide synchronization of processes, network virtual addressing, software transparent hardware distribution, communication at the intermodule level, and data security.

5.6.2.3 C.mmp A Multi-Mini-Processor - The C.mmp at Carnegie-Mellon connects as many as 16 processors to a common memory system by using a cross-bar switch.[35] Except for this cross-bar switch and the particular method of distributed system-wide interrupts, there is little to differentiate the C.mmp architecture from that of a multiprocessor. While the cross-bar switch network put it in the LNS class, the presence of the singly existing operating system, HYDRA,[36] makes the C.mmp also fit the class of multiprocessor. Actually, the C.mmp is a crossbreed between the LNS and the multiprocessor.

5.6.3 Multiprocessors - The most generally accepted definition of a multiprocessor is a system with more than one central processor, sharing a common system memory, and operated under the control of a single set of executive software. Any of the processors is interchangeable with any other and capable of

[35] D.R. Anderson, "The EPIC-DPS -- A Distributed Network Experiment," EASCON '75 Record (September 1975), pp. 121-A - 121-G.

[36] W.A. Wulf and C.G. Bell, "C.mmp - A Multi-Mini Processor," AFIPS FJCC, XLI, Part 2 (1972), pp. 765-778.

[37] W.A. Wulf et al, "HYDRA: The Kernel of a Multiprocessor Operating System," CACM, XVII:6 (June 1974), pp. 337-345.

executing either task or executive programs. Any central processor can access any part of the common memory shared by all the central processors. Multiple central processors provide for increased reliability and fail-soft capabilities at the expense of additional hardware.

Multiprocessing is a well known, well used technique in the computer industry, compared with other categories of distributed systems. (This is borne out by the presence of ample software support for the multiprocessing systems.) Current multiprocessor systems are available in both military and commercial forms and are supported by operating systems and operational software.

5.6.3.1 Military Multiprocessors - Multiprocessing systems currently present in government inventory are used in avionics, air traffic control, and shipboard applications. In production from three to five years, the systems listed below are typical.

AN/UYK-7 - A general purpose multiprocessor system design for shipboard use. Uses a maximum of 262,144 words memory, 32 bits each. The maximum configuration of the AN/UYK-7 is a 3X4 (3 CPU, 4 IOC).

ARTS-III - A general purpose, 30-bit, multiprocessor system currently being used for air traffic control. The maximum configuration is eight processors and sixteen 16K memory modules.

S-3A - The Sperry Univac 1832 Multiprocessor system is being used in the S-3A Viking ASW aircraft. This system is a fixed configuration machine (2CPU, 2IOC), mechanically designed for use in the Viking aircraft.

5.6.3.2 Commercial Multiprocessors - Numerous multiprocessor systems are available commercially. These systems offer a full range of performance within the medium to large ranges and are supported with comprehensive operating system capabilities and systems program libraries that include compilers, assemblers, development tools, and data base management. Sperry Univac is the manufacturer perhaps most committed to multiprocessor products. IBM, Burroughs, and Digital Equipment Corporation are much less committed, but have developed multiprocessor products.

As mentioned previously, the software available for commercial multiprocessor products is extensive. As a typical example, consider what is available for the Univac 1100 Series:

- Operating system
- Language processors - assemblers, COBOL, FORTRAN, ALGOL, etc.
- Data base/data communication software
- Other utility programs

5.6.4 Distributed Function Systems (DFS) - In this fourth category of distributed systems, the processing capabilities of the system are distributed on a functional basis. Autonomous processing units can operate simultaneously while sharing a common virtual memory. These autonomous units should have sufficient logic, registers, control, and other features to perform tasks without being under the control of a conventional CPU.

5.6.4.1 SYMBOL - The SYMBOL system has eight different specialized processors, each of which operates as an autonomous unit. Each unit is linked to the system by the main bus and communicates with other units through shared storage space, enabled by a paging virtual address scheme. Translation of virtual addresses to absolute is handled by the memory controller, which also handles storage management. All of the processors take jobs from individual job queues, which are maintained by the job controller. The overall philosophy of SYMBOL was to reexamine the traditional stages of software reduction and execution, and where possible, dedicate hardware processors to these functions.

5.6.5 Distributed Element Systems (DES) - Any computer with a single global control unit that drives multiple processing units, all of which either execute or ignore the current instruction, falls into the category of distributed element systems. This category can be further divided into associative processors, parallel processors, and ensembles. Any DES machine in which the processing units (or processing memory) are addressed by a property of the data contents rather than by address are classified as associative processors. Parallel processors are those in which the processing elements have the same order of complexity as current small computers and which typically have a high level of interconnection between processing elements. Parallel processors in which the interconnection level between processing elements is very low or nonexistent are ensembles.

Distributed element systems have many advantages. They offer reliable configurations with the ability to degrade gracefully. They are most effective for problems with large amounts of parallelism. Duplicate structures result in lower costs.

Finally, software for large systems is easier to construct, and therefore less executive function is required. The deficiencies include a lack of flexibility, the possibility for I/O bottlenecks, and limitations in the types of problems that can be processed on these systems.

5.6.5.1 STARAN - The STARAN processor is an associative processor DES machine. Its distinguishing features are a multidimensional access array memory, a content addressable memory, a simple processing unit at each memory word, and a unique permutation network for shifting and rearranging data in memory. STARAN is most effective when used in applications requiring very fast processing, highly dynamic data, large numbers of data items requiring similar processing and an immediate response to certain inquiries. On the other hand, STARAN has the disadvantage of not being compatible with peripherals designed for sequential machines.

5.6.5.2 ILLIAC IV - ILLIAC IV has a four-nearest-neighbor interconnection structure. In order to sustain the rate of instruction flow to the processing elements, each control unit is capable of parallel indexing, instruction fetch, and local arithmetic. Each processing element has a 2048 word memory. ILLIAC IV is especially efficient in applications which require a great quantity of data to be exchanged among neighboring processing elements, weather forecasting, for example. Unlike associative processors, ILLIAC is amenable to implementation out of microprocessors. A disadvantage of the ILLIAC IV architecture is that the close association of each piece of memory with its own processing element means that all memory is not equally accessible from every processing element.

5.6.5.3 PEPE - PEPE (Parallel Element Processing Ensemble) is an ensemble designed for the ballistic missile radar defense processing. Operated under the control of a CDC 7600 host, it is associatively organized, highly parallel, and capable of executing three instruction streams simultaneously: correlation, arithmetic, and associative output. PEPE's processing elements were designed for applications that do not require direct interconnection, thereby providing a reliable structure in that any processing element can substitute for any other processing element.

5.6.6 RCSDF Requirements in Distributed Systems - Each of the RCSDF hardware elements has different processing capabilities (e.g., general emulation for the QM-1) and processing speeds. To facilitate communications between various elements, techniques for controlling these hardware elements are needed for the emulation procedures on the RCSDF.

RCSDF is thought to have the capability of emulating existing systems. This includes the distributed systems organization. To provide the promised emulation efficiency and expertise, RCSDF will need to anticipate the user's system requirements and to reconfigure the RCSDF hardware elements into a configuration that resembles the users' target system. The RCSDF relationships with distributed systems organization translate into many areas that require further understanding.

RCSDF Hardware Elements Characteristics - The functional characteristics of each existing RCSDF component and each RCSDF component to be procured have to be fully understood. The mechanism by which these RCSDF components will be connected should also be fully understood. Another important consideration is the data transfer rate of each data path. For components with large processing bandwidths like STARAN and the microprocessor array, enormous data transfer rates are required for accessing data in mass storage, and matching the transfer bandwidth with processing bandwidth becomes a major problem.

User System Design Spectrum - RCSDF needs to anticipate the system architectures that are likely to be designed by the users, then specify a set of representative architectures that would cover the entire spectrum of user systems designs, and finally determine the role of each RCSDF hardware element in the emulation of each representative architecture.

User Application Spectrum - To specify the set of representative architectures, the user applications would also play a very important role. When the spectrum of the user applications is determined, then the required processing capability (e.g., associative processing and list search/merge), processing bandwidth, and data transfer bandwidth can be assessed. Subsequently, the necessity of hardware additions to RCSDF can be evaluated.

Emulation Control Structure - Although each RCSDF component should have its own operating system, the collection of components also needs a centralized operating system, coordinating communications among various components. To achieve this, we have to be able to define the system building blocks, the functional units. In addition standards need to be set to interface various functional units to coordinate concurrent processes and to facilitate resource sharing.

5.6.7 RCSDF Distributive Systems Organization/Recommendations/ Conclusions

- RCSDF components are most suitable for system architectures in which components are loosely coupled, distributed system network architectures, for example. Properly decomposed, the facility emulation content could be adaptable to representing different distributed architecture forms. However, doing this requires a deep understanding of control structure and architecture, areas which need further study. Paragraph 5.6.6 discussed knowledge which needs to be established soon.

Looking further into the future, the problems of interconnecting the facility components and of controlling the interconnected elements should be resolved so that timing is not interlocked between components yet concurrency is maintained. The emulating system will inherently be a distributed system of computers but must be made to appear to the user as a different architecture through emulation. In some cases, features or capabilities of the emulating system may be explicit (available to the user) or implicit (embedded within the control software and masked from the user), depending on the architecture to be emulated.

The key to providing the explicit/implicit capabilities is the proper decomposition of the emulating system so the explicit/implicit boundary can be easily constructed as a function of user emulation event. The initial step toward this goal is to define the emulating system architecture and the emulation control structure. It is recommended that this proceed immediately to provide a foundation for developing further emulation and measuring skills.

The degree of autonomous intelligence within the facility software for resource management, process regulation, and, in general, control of the emulation process is a subject that should be addressed in.detail. Self control should be incorporated into the network in order to free the user from explicit control of network operations. Analogously, support of implicit capabilities will require a larger degree of intelligence within the facility software, hardware, or firmware than support of the explicit approach.

The efficiency of a multicomputer distributed architecture for supporting multiprocessor architecture with tight synchronization will be considered ultimately and should not be a concern in the near term. Current objectives should emphasize technical feasibility and the development of techniques. Emulating system hardware features can be developed to improve emulating system efficiency should improved operations be desired later. Efficiency should not be considered in the near future.

5.7 Design Languages Technical Baseline

The purpose for this baseline study is to 1) identify those languages which have been defined for system specification, design, and development, and 2) evaluate their applicability for the RCSDF environment and their usefulness in aiding system design and development.

5.7.1 Language Identification and Evaluation - The languages are categorized as three basic types: 1) requirement specification languages, 2) high order languages (software design languages), and 3) emulation design languages (hardware design languages). Each of them is discussed below.

5.7.1.1 Requirement Specification Languages - A cost effective development of systems necessitates a carefully controlled system requirements specification phase. What is needed is a language which enforces a discipline on the human requirements engineer and additionally provides a specification that is machine-processable. Automated language techniques should be judged on such criteria as 1) the ability of management to see the progress of the system development through its life cycle (traceability); 2) the ability to detect errors, ambiguity, inconsistencies, and incompleteness in system description (validation); 3) the ease of modifying parts of the system description without introducing errors (configuration control); and 4) the degree to which a system specification may be translated into a detailed design (implementation).

5.7.1.1.1 Existing Languages - Numerous methodologies have recently been touted as a potential solution to the problems and resulting high cost of software engineering. Among the systems which have been devised to implement these methodologies are the Hierarchy and Input/Output Process System (HIPO) developed by IBM,[37] LOGOS developed by Case Western Research,[38] Information System Design and Optimization System (ISDOS) developed at the University of Michigan,[39] and the Software Requirements Engineering Program (SREP) developed for the Ballistic Missile Defense Advanced Technology Center (BMDATC) by TRW.[40]

[38] IBM, "HIPO: Design Aid and Documentation Tool," SR20-9413-0 (1973).

[39] C.W. Rose, "LOGOS and the Software Engineering," AFIPS Proceedings, FJCC XLI (1972), pp. 311-323.

[40] D. Teichroew and M. Bastarache, "PSL User's Manual," ISDOS Working Paper No. 98 (1975).

[41] C.G. Davis and C.R. Vick, "The Software Development System," Proc. Second International Sofware Eng. Conf. (October 1976).

The heart of any requirement/design language which is machine-processable is a competent data base manager and tools for design retention. As a result, we have chosen to describe two of the systems and the associated languages we feel have come closest to this ideal.

5.7.1.1.1.1 ISDOS/Problem Statement Language - The ISDOS model divides each application into three parts: 1) the environment in which the application occurs, 2) the proposed or "target" system to accomplish the application, and 3) the project responsible for the application throughout its life cycle. The target system receives inputs from an interface with the environment, maintains or updates internally used data (entities), and generates outputs to an interface with the environment. The Problem Statement Language (PSL) objects are interconnected by PSL relationships. Relationships between objects may be decomposed into smaller, more detailed pieces until the application requirements are completely specified. The objects may have properties and each property may have a property value. All of this information is stored in a relational data base.

A set of software tools, called the Problem Statement Analyzer (PSA), is used to enter the system requirements into the data base. PSA is composed of three main parts: a command language for entering new data and modifying existing data in the data base, an analyzer to syntactically check each PSL statement, and a report generation facility to produce reports on various aspects of the data base.

5.7.1.1.1.2 REVS/Requirements Statement Language - BMDATC's SREP has developed REVS, Requirements Engineering Validation System. The REVS system model expresses functional requirements in terms of stimulus-response requirements networks. A flow oriented approach is used to state requirements, and performance can be checked for required timing and information accuracy.

The requirements networks (R-NETs) are developed in a top down fashion. The basic structure is the ALPHA which has one input and one output. An ALPHA can be decomposed into a SUBNET of lower level ALPHAs. The flow structure of a net contains sequencing information. ALPHAs may be executed in series or in parallel , as indicated in the type of node connection. A SUBNET or ALPHA can be executed a given number of times. The ALPHAs are the bottom level of description and contain executable descriptions (in PASCAL).

A Requirement Statement Language (RSL) is based upon four primitive objects:

1) elements

2) relationships

3) attributes

4) structures

Elements are the "nouns" of RSL. Relationships are the RSL "verbs" and are the links between the RSL elements. Attributes are the RSL "adjectives". Each element has exactly one value for each of its attributes. Structures are the RSL representation of the nodes described in the preceding paragraph.

RSL is extensible in the sense that new elements, relationships, and attributes may be created in terms of previously defined RSL objects. The RSL statements that comprise a requirements specification are stored in a relational data base.

5.7.1.1.2 Traceability - Traceability allows project management to monitor system development through the system life cycle. Such monitoring is essential to insure that the user or customer needs are accomplished by the system under development. In general, there is no one-to-one correspondence between the statement of user requirements and the physical design. Of the two design methodologies represented here, only the SREP methodology claims traceability in design. However, this claim has not been substantiated.

5.7.1.1.3 Validation - A requirements language should contain a data base management system which lends itself to the static and dynamic analysis of requirements specifications. Static analysis refers to techniques for verifying the completeness, consistency, and correctness. Dynamic analysis refers to techniques for verifying the dynamic performance by simulation. Both PSL and RSL satisfy this second criteria to some extent.

5.7.1.1.4 Configuration Control - Configuration control data enables management to take a "snapshot" of the requirements description at each stage of its development. Thus, management can see how the description evolves. In order to accomplish this, each object in the description must have a descriptor associated with it. The descriptor might indicate when the object was entered into the data base as well as subsequent alterations to it. Both PSL and RSL associate the name of a person with each object in the requirements specification.

5.7.1.1.5 Implementation - The final criterion for a requirements language is that it aids the process of translating requirements into greater design detail. If the initial system requirements specification can not be met by the detailed design, the requirements specification should be revised, implying the need for interpretation of the requirements language as the detailed design is submitted to the machine process. RSL does provide the framework for this interpretation by allowing detailed design information in the form of PASCAL statements with ALPHAs.

5.7.1.2 Higher Order Languages (HOL) - A HOL is a software design language that assumes the proportions similar or equivalent to that represented by a COBOL, JOVIAL, or FORTRAN programming language. The discussions contained in the following paragraphs summarize the results of a recent survey of currently available HOLs.

5.7.1.2.1 Standardized HOLs - In January 1975, a DoD High Order Language Working Group (HOLWG) was chartered to investigate programming language requirements. The HOLWG identified a need to evaluate existing approaches and to recommend adoption or implementation of the desirable characteristics as a common language. The selected HOL is to be used for the development of software for what are called embedded computer applications (i.e., command and control, communications, avionics, shipboards test equipment).

The general requirements are summarized as follows:

- Simplicity
- Reliability
- Readability
- Maintainability
- Efficiency
- Implementability
- Machine Independence
- Portability
- Definition

A sample of the languages selected for analysis by the HOLWG are categorized and described in the following paragraphs, together with a summary of the findings and recommendations made by the evaluators and comments relating to the RCSDF.

PL/1 - This language adopts its expression syntax from FORTRAN, its block structure from ALGOL 60, and its data description facilities from COBOL. It is rich in data types, data attributes, control structures, and input-output features. However, it is not strongly typed and lacks extensibility, parallel processing, synchronization, and real-time features. Evaluation indicates that the language should be considered as a potential base language.

PASCAL - This language provides richer declarative facilities than ALGOL 60. It is also strongly typed, well designed language with extensibility and control structure facilities. It can be used as a starting point for designing new languages. However, it does not support parallel processing, real-time, error handling, and precision facilities. The results of the language evaluation suggest that PASCAL could serve as the common language because relatively few features need to be added.

ALGOL 68 - This language is extensible, providing great user flexibility. It is a block-structured language providing a rich collection of data types and facilities. In addition, it allows for the definition, control, and synchronization of parallel processes. It is a clearly designed language that can be modified easily. The language was most consistently recommended by its evaluators as the best candidate for the base language.

FORTRAN - Essentially the first higher level language, FORTRAN lacks data description facilities, control structures, pointer variables, and recursive procedures. Evaluation results indicate that FORTRAN is not suitable as a base language because its design does not reflect advances in language design.

COBOL - Its data description facilities and its concept of environment, data,and procedure divisions are still important today but the algorithm specification part of COBOL is behind the current state of the art. Evaluation results indicate that COBOL is not suitable as a base language for embedded computer applications, both because it was not designed for this class of applications and because it is dominated as a base language by more recently designed languages.

CMS-2 - This language now contains most of the structured programming features common to modern HOLs as well as features considered necessary for embedded military systems. The results of the evaluation indicate that the language is not acceptable as a base language because it lacks too many of the required features.

The languages discussed below are more specific than the preceding six.

HAL/S - This language is designed especially for use on airborne computers and other embedded systems in connection with the space shuttle program. It has a real-time processing facility with a flexible facility for scheduled processes in a process queue. Execution and storage allocation for scheduled processes are left to the operating system. Evaluation results suggest that the language could be considered as the base language.

SPL/1 - Originally designed for application to acoustical signal processing embedded computer, SPL-1 is a block structured language that allows users to express functions and control in a structured form. It provides facilities that permit users to create, define, delete, and synchronize control of parallel processes. Like HAL/S it has the capability of establishing the generation of processes with the resources dynamically allocated at execution time by the operating system. It was recommended as a base language candidate.

5.7.1.2.2 Multiple High Order Languages - The availability of a single HOL that will be accepted by all users of the RCSDF is ideal but probably unrealizable. A current system developed at Sperry Univac, called the Translator Writing System (TWS), was designed to facilitate the construction of high-level compilers by supplying the nucleus of a compiler translator, called the compiler skeleton. It consists of a general purpose lexical analyzer and a table driven parser. The parse tables are constructed by a grammar analyzer when supplied with a BNF description of the source language. The semantic routines are provided by TWS users for each HOL. The routines produce the intermediate language code that becomes the input to the machine code generator.

A common Intermediate Exchange Language (IEL) will need to be developed to serve as input to the machine code generator. Theoretically, once a code generator has been developed for a target machine, it can compile for any HOL source accepted by the TWS.

Future efforts will use the automation concepts found in the TWS to simplify the development of code generators. Hypothetically, a machine description (e.g., ISP and SMITE) would be processed and tables constructed to control a code generator driven by IEL input. This concept was introduced in Section 2 and called Auto Code Generation (ACG). Similar efforts are being pursued throughout the industry. The TWS and ASG techniques should be defined for the unique requirements of RCSDF. Examples include the ability to generate code for parallel processors, enforce structured processes, and permit dynamic resource allocation.

5.7.1.2.3 HOL Summary - The HOLWG concluded that no single existing standard language satisfied all the language requirements even though all the capabilities were available in existing languages. The group recommended that work toward the production of a single HOL should start with an existing base language. PL/1, PASCAL, and ALGOL 68 were recommended as suitable base languages. Current trends in HOL development indicate that additional capabilities are being included that provide users with high level interface to operating systems for resource and task control facilities. The RCSDF development project must consider the characteristics and intended functional capabilities of these interfaces and evaluate them on the basis of the requirements for emulation on the RCSDF.

5.7.1.3 Emulation Design Languages - A truly high level language whose sole purpose is emulation implementation does not exist. Nevertheless, Emulation Design Language (EDL) requirements can be satisfied by some of the Computer Hardware Description Languages (CHDLs).

5.7.1.3.1 CHDL Description Levels - Computer hardware systems can be described in four levels of abstraction. Each level of description carries an added amount of implementation detail than the level preceding it. Conversely, a broader view of the architecture can be obtained more readily from a higher level description. The four levels are described below.

The Circuit Level - The components used here are Rs, Ls, Cs, voltage sources, and nonlinear devices. Normally, groups of components are used repetitively, and each group of components becomes a good candidate for a primitive component for the next higher description level.

The Logic Level - The logic level description applies only to digital devices. Components such as AND, OR, and NOT gates are used at this level. Description in the logic level eliminates much unnecessary detail and allows behavior to be traced on circuits that could be extremely complex at the circuit level description.

The Register Transfer Level - The next higher level has been identified as the Register-Transfer (RT) level. The components are the registers, data operators, and transfer operations. Data operations are defined by groups of logic level components. Clocks and some conditional rules are used to transfer data from register to register. It is at this level that most digit components are described. The RT level is closely related to microprogramming ,in that each RT statement could conceivably be transformed into a microcode or a set of microcodes.

The PMS Level - This extra level of description is used for an ensemble of computing devices (e.g., processors, memories, I/O devices, communication lines). At this level, internal details are omitted, leaving only overall information about each device, for example, cost and performance criteria.

5.7.1.3.2 CHDL Survey - In 1973, a Consensus Language (CONLAN) working group was set up to specify and develop a universal consensus language for all phases of hardware design. So far, they have defined CONLAN as an application oriented family of sublanguages (SUBCONLANs) and have identified a universal base language (BASE CONLAN) from which all SUBCONLANs can be derived in a formal and consistent way. The working group is expected to complete the specification of CONLAN in the near future. Until this is completed, we can only survey some of the existing CHDLs.

AHPL - A hardware description language, AHPL is based on the notational conventions of APL.[41] Bit array operations are allowed, and most of the rich operator set of APL is retained. A few extensions were also added. AHPL is most suitable for the logic level descriptions, although it may be quite easily extended to include RT level descriptions. An AHPL compiler exists at the University of Arizona, where AHPL descriptions of a combinational logic unit can be translated into a wiring list and a fan-in list.

CDL - Structured like ALGOL, CDL is normally used for RT level descriptions. Like most RT level languages, it identifies both control and data carrier description capabilities, but allows subcarriers of such carriers to be declared. In the control portion, CDL also allows concurrent activities in each step. It uses special control variables to form a label describing the conditions for execution of a control step. A CDL simulator/ translator, available since 1967, translates CDL programs into Boolean equations for each input device.

ISP - Instruction Set Processor (ISP) is an ALGOL-like language, which uses block structures to describe various subsystems of the system being discussed. It has the capability to handle concurrency and activity sequencing in a simple fashion; it also provides an adequate set of data and control operators. An ISP description consists of a list of declarations followed by the processes and action sequences. In the declarations, hierarchy of register arrays, registers, and subregisters can be defined. ISP is a procedural language, as opposed to CDL and AHPL. Parallel actions are grouped into time blocks, and

[42] F.J. Hill and G.R. Peterson, Digital Systems: Hardware Organization and Design (1973).

sequential actions are described as lists of time blocks. This definition can be used recursively to build very complex time blocks. Conditional statements are used by ISP to select actions under a condition described by a Boolean expression.

The ISP compiler at Carnegie-Mellon University can produce a symbol table, a statement table, and a set of subroutine calls from the ISP program. The information produced can be used in a simulator or a design verifier and also in a design automation program.

PMS - PMS (Processor Memory Switch) is used to describe the physical structures of various subsystems of a computer and how they are connected with each other. In the PMS level, the fine structure of information processing is ignored and the transfer of information among the system components is considered. It is most suitable for descriptions at the PMS level.

Because of the compactness of information offered by the PMS descriptions of systems, the comparison of various systems can be simplified. However, PMS cannot be easily used as a language form, one of its main disadvantages.

5.7.2 RCSDF Design Language Requirements - An RCSDF implementation should require that the system requirement specification also be expressed in a language compatible with both software and hardware design languages. This will permit host subsystem validation tools to be developed to monitor the design process.

Capabilities found in software design languages establish a number of requirements for the RCSDF architecture, assuming the languages are used to define the application software. A sample list is shown below:

- o Parallel control structures
- o Parallel control path implementation
- o Process mutual exclusion
- o Separately compiled segments
- o Minimal built-in control structures
- o Data type specification - shared/non-shared
- o Dynamic allocation

While the above language capabilities are compatible with the RCSDF usage methodology, other capabilities are less compatible and will force special requirements on the RCSDF development cycle. Examples of such capabilities are:

- o Machine configuration constants declaration
- o Built-in I/O facilities
- o Resource status interrogation (and control) facilities

The HOLs utilized in the TSDF must permit structured system development and integration of CHDL produced functions. Specifically, they must permit clearly defined interfaces to be established between HOL produced functional modules (processes) and EDL produced functional modules, and between the code (micro- or macro-) and the operating system. It must not place restrictions on the management of resources.

Since deployable systems designed for testing on the RCSDF are expected to consist of processes of macrocode sequences generated by a HOL compiler and microcode sequences generated by an EDL compiler, it is desirable that the language used to construct either type of process be consistent, possibly one a subset of the other.

The complexity of the RCSDF system and its usage requires that an organized system definition exist to describe the structure and capabilities of the deployable system to be emulated. Because of the autonomous nature of the RCSDF components required for emulation, a hardware description language with the abstraction level of PMS appears necessary to describe the deployable systems operational characteristics. For intercomponent communications protocol, a capability for describing asynchronous, parallel, operations is also needed. The RT languages appear sufficient to describe those communication sequences. The RT language is also seen as the language level required to describe the deployable system's computer functional architecture (repertoire). However, since most RT languages are currently used only for describing mainframe computers, some extensions are anticipated.

Ultimately, performance measurement is the chief requirement of the RCSDF. It is desirable, during emulation, not to have to change the application software to facilitate performance data extraction. Thus a dialect of the general Emulation Design Language (EDL) must be available to describe the data extraction process. By utilizing the EDL, it is possible to extract the performance data required in parallel with the actual emulation and minimize system overhead.

5.7.3 RCSDF Design Language Recommendations and Conclusions - It is important that a system model be developed for RCSDF. The model must be reflected in the requirement language utilized to specify application requirements and be compatible with capabilities available in the hardware/software design languages. To achieve this, the requirements language must be translated into a data base which will support various validation tools, including management tools to chart progress through the life cycle by means of traceability and configuration control.

The efforts of the HOLWG and other groups shall continue to affect the HOLs which can be used for software development. If in the future all DoD systems are to be programmed using a common HOL, then RCSDF users will be using this language exclusively at some time in the future. Thus it becomes imperative that the RCSDF project maintain close rapport with this program so that when unique requirements are identified from new system architectures, they can be proposed for inclusion in the common DoD HOL.

From the discussions in paragraph 5.7.2, it can be seen that a CHDL with description levels of RT and PMS is desirable for RCSDF. However, because of the immaturity of this technology, use of CHDLs should be restricted to emulation code design, i.e., the use of an EDL. An EDL translator or compiler will probably not be available until later. However, an EDL should be used to formulate the deployable system architecture and where possible, CHDLs used to describe the existing RCSDF hardware components.

6. Conclusions

In general the Sperry Univac study team has found RADC's concept of Total System Design (TSD), utilizing a Reconfigurable Computer System Design Facility (RCSDF) for system emulation, to be a viable method for reducing total system hardware and software costs. Two primary reasons can be cited that lead us to this conclusion. First, technology is supporting the availability of low cost, specialized hardware elements, thus permitting selection of hardware elements tailored to a particular military application requirement thereby reducing software costs. Such selection of hardware elements, however, must be delayed until the system design requirements are fully known and shown to be reliable and viable. In addition, the fact that these specialized hardware elements are available will reduce hardware design procurement expenditures. Secondly, one of the results of the TSD concept will be the establishment of system component interface standards, design and documentation standards, and system performance and validation procedures, which will increase the availability of viable hardware and software system elements and increase competition during system procurements. Development of the RCSDF emulation facility as an integral part of a total system design methodology would promote both inter- and intra-system standardization as well as more reliable procurement procedures.

The study team recognizes the technical risks involved in attempting to provide a facility capable of emulating a variety of system architectures (see paragraph 2.5.2). To reduce the risks, while promoting the benefits, Sperry Univac suggests two alternatives to the four year development plan described in Section 4. The alternatives (Figures 6.1b and 6.1c) would proceed under a phased development approach permitting risks to be re-evaluated and benefits identified before starting the next phase. (Recall

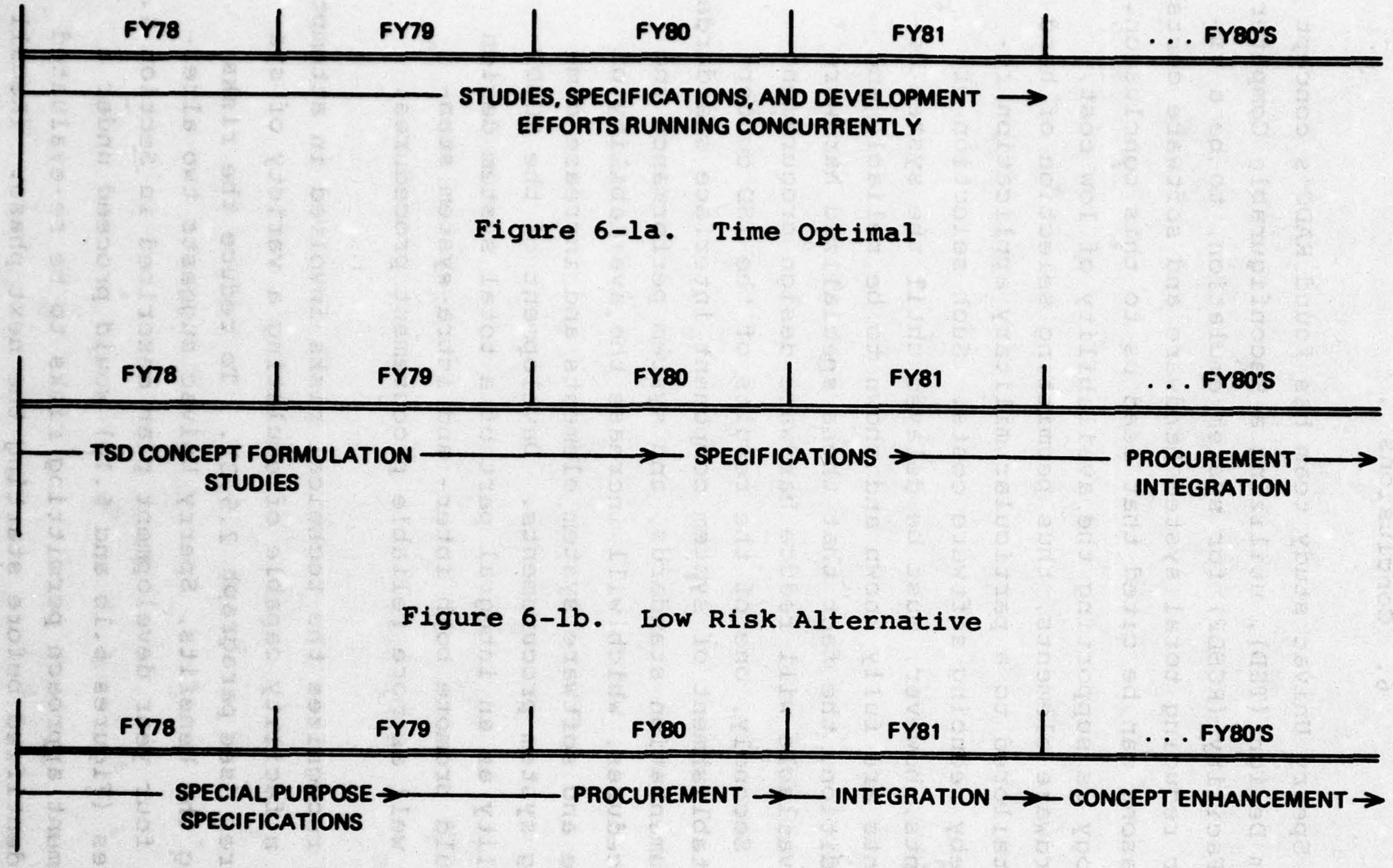


Figure 6-1a. Time Optimal

Figure 6-1b. Low Risk Alternative

Figure 6-1c. Case Study Driven Alternative

Figure 6-1. RCSDF Development Plan Time Lines

that the tasks described in paragraph 4.3 were designed for a plan to achieve a general purpose emulation capability which was a contractual objective.)

The first alternative completes all TSD concept formulation studies before initiating implementation/specification tasks. The primary objective would remain to develop a general purpose emulation facility, i.e., RCSDF. The advantage would be a reduction in technical uncertainty and hence unforeseen cost factors. The disadvantage would be a longer unit of time before total system design benefits and feasibility could be demonstrated.

The second alternative would narrow the scope of the initial RCSDF development by reducing its dependency upon the evolving total system design methodology. This could be achieved by identifying a single case study and developing the facility to meet the emulation needs associated with the required system or subsystem. Existing tools (e.g., languages and language translators) would be used even though they might not be able to provide the ultimately achievable benefits. New languages and tools required (e.g., scenario generators and performance query languages) would be developed to meet just the requirements of the selected case study. The advantages of this alternative are less time to demonstrate benefits and a working knowledge of the problems which remain. The disadvantage is the potential higher program cost associated with hardware and software development efforts which fail to meet objectives of other case studies/applications and hence must be discarded. It should be noted that similar military system development costs could be removed/reduced should the RCSDF become a proven tool for total system design.

Figures 6.1a through 6.1c summarize the contracted development plan and the two suggested alternatives. The Sperry Univac study team has concluded that the tasks identified in the contracted development plan (Section 4),tailored to meet specific case study objectives (alternative 2 - Figure 6.1c), would ultimately provide RADC with the most timely benefits at a lower risk and cost.

REFERENCES

Anderson, D. R. "The EPIC-DPS -- A Distributed Network Experiment." EASCON '75 Record (September 1975), pp. 121-A - 121-G.

Batcher, K. E. "The Flip Network in STARAN." Proc. 1976 International Conf. on Parallel Processing (August 1976), pp. 65-71.

Batcher, K. E. "Sorting Networks and Their Application." Proc. SJCC (1968), pp. 307-314.

Batcher, K. E. "STARAN Parallel Processor System Hardware." AFIPS NCC XLIII (1974), pp. 405-410

Bayer, D. L., and Lycklama, H. "MERT - A Multi-Environment Real-Time Operating System." ACM Proc. of Fifth Symp., O. S. Princ. (1975), pp. 33-42.

Bell, G., and Newell, A. Computer Structures: Readings and Examples. New York: McGraw-Hill, 1971.

BMD Advanced Technology Center. "BMDATC Software Development System - Program Overview." I (July 1975).

Davis, C. G., and Vick, C. R. "The Software Development System.' Proc. Second International Software Eng. Conf. (October 1976).

Denning, P. J. "Virtual Memory." Computing Surveys (September 1970), pp. 153-189.

Dijkstra, E. W. "Cooperating Sequential Processing" in Programming Languages, Genuys, ed. New York: Academic Press, 1968. pp. 43-110.

Dijkstra, E. W. "The Structure of the Multiprogramming System." Comm. of ACM XI:5 (May 1968), pp. 341-346.

Farber, D. J. et al, "The Distributed Computer System," Proceedings IEEE Computer Society International Conference (March 1973), pp. 31-34.

Hansen, P. B. "The Nucleus of a Multiprogramming System." Comm. of ACM XIII:4 (April 1970), pp. 238-242.

Hansen, P. B. Operating System Principles (1973).

Hill, F. J., and Peterson, G. R. Digital Systems: Hardware Organization and Design. New York: Wiley, 1973.

Horning, J. J., and Randell, B. "Process Structuring." Comp. Surveys V:1 (March 1973), pp. 5-30.

"Hyperdimensional Microporcessor Collection Seen Functioning as Mainframe." Digital Design (November 1975).

IBM. "HIPO: Design Aid and Documentation Tool." SR20-9413-0 (1973).

Kinney, L. L., and Christiansen, B. P. "Functional Requirements of a Memory Manager." Univac PX 1174 (May 1976).

Koczela, L. J. "The Distribution Processor Organization" in Advances in Computers. New York: Associated Press, 1968.

Lawrie, D. H. "Access and Alignment of Data in an Array Processor." IEEE Trans. on Computers (December 1974), pp. 1145-1155.

Parnas, D. "On the Criteria To Be Used in Decomposing Systems into Modules." *Comm. of ACM* XV:12 (December 1972).

Pease, M. C. "The Indirect Binary n-Cube Microprocessor." *IEEE Computer Society Repository* 75-100 (1975).

Rose, C. W. "LOGOS and the Software Engineering." *AFIPS Proceedings, FJCC* XLI (1972), pp. 311-323.

Stone, H. S. "Parallel Processing with the Perfect Shuffle." *IEEE Trans. on Computers* (February 1971), pp. 153-161.

Thurber, K. J. "Associative and Parallel Processors." *Computing Surveys* (December 1975).

Thurber, K. J. et al. "A Systematic Approach to the Design of Digital Bussing Structures." *Proc. Fall Joint Computer Conf*. (1972), pp. 719-740.

Teichroew, D., and Bastasache, M. "PSL User's Manual." ISDOS Working Paper No. 98 (1975).

Widdoes, L. C. "The Minerva Multi-Microprocessor." *Proc. Third Annual Symp. on Computer Arch*. (1976), pp. 34-39.

Wilhelm, N., Pessel, D., and Merriam, C. "The CERF Computer System." *Proc. of the NCC* (1976), pp. 765-768.

Wulf, W. A. and Bell, C. G. "C.mmp - A Multi-Mini Processor," *AFIPS FJCC*, XLI, Part 2 (1972), pp. 765-778.

Wulf, W. et al. "HYDRA: The Kernel of a Multiprocessor Operating System." *Comm. of ACM* XVII (June 1974), pp. 337-345.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

AMERICAN REVOLUTION BICENTENNIAL 1776-1976